# Heuristically Inducing A Distance Metric From User Preferences For Clustering

*Xiaojin Zhu*
LTI/CALD
Carnegie Mellon University
zhuxj@cs.cmu.edu

*Tianjiao Chu*
Philosophy/CALD
Carnegie Mellon University
tchu+@andrew.cmu.edu

*Jie Zhu*
LTI
Carnegie Mellon University
jzhu@cs.cmu.edu

*Rich Caruana*
CALD
Carnegie Mellon University
caruana@cs.cmu.edu

May 3, 2001

**Abstract**

Traditional clustering algorithms cannot incorporate domain expert's preferences into clustering process. Since one major goal of clustering is for the experts to gain insight into a dataset, it makes sense to allow experts to express opinions to guide clustering in some ways. We propose a mechanism for the experts to express a small number of clustering preferences that they wish to be true. We devise a set of heuristic algorithms which induce an appropriate distance metric from the expert preferences, such that clustering under the induced metric maximally satisfies the expert preferences, while remains good in traditional measures. Experiments show promising results.

## 1 Introduction

Traditional clustering algorithms operates without human intervention. However in many cases domain experts (e.g. astrophysicists, biologists, etc.) have certain prior domain knowledge, and they would like to express the knowledge to guide the clustering in certain ways. Imagine a domain expert, besides giving the dataset to the computer, also specifies some preferences regarding the relations among a few data points in the dataset. Note "a few" is the key word: the dataset is usually too large to be examined completely by human. The expert may say the following:

- "data point A looks like to be in the same cluster as data point B"

- "I'm pretty sure C and D are in different clusters"
- "I think E is much closer/similar to F than G is to H"

The clustering problem now becomes one of finding a good clustering that also mostly satisfies the preferences set forth by human experts.

Our approach is to induce an appropriate distance metric that tries to take the preferences into account, and then cluster the data with the induced metric. It has long been recognized that different distance metrics on the same dataset, with the same clustering algorithm may produce different clustering results [6]. For example, many clustering programs assume a default Euclidean distance in the raw data space. In this case, scaling axis (which corresponds to feature weighting) induces different Euclidean distance metrics and can affect clustering result. Let's assume data points are represented as $k$ dimensional vectors of continuous attributes, e.g., $A = (a_1, a_2, \ldots, a_k)$. Under this condition, the default distance metric between two data points is the Euclidean distance: $dist(A, B) = \sqrt{\sum_{i=1}^{k}(a_i - b_i)^2}$. This assumes that the dataset has been properly prepared by the domain expert, so that each attribute has appropriate scale and the distance makes sense. However imaginably this can go wrong. For instance consider a health care dataset containing two attributes 'weight' and 'height' in $kg$ and $cm$ respectively:

person1 (50, 160) person2 (55, 165) person3 (51, 190)

Common sense tells us that person 1 and person 2 are more or less normal, while person 3 is abnormal. The Euclidean distance coincides with our intuition since e.g. $dist(person1, person2) = 7.07 < 30.02 = dist(person1, person3)$. But if the height is in $m$ (meter):

person1 (50, 1.60) person2 (55, 1.65) person3 (51, 1.90)

Now $dist(person1, person2) = 5.00 > 1.04 = dist(person1, person3)$, i.e. Euclidean distance no longer matches our intuition. In this case different units on an attribute result in different distance measures, and may give rise to unexpected clustering to a health care expert. Note normalizing individual attributes or performing principal component analysis on the dataset can't solve this problem. We believe this situation will often occur, since it is very hard for the domain expert to choose appropriate attribute scales. Nevertheless, it should be relatively easy for the expert to give a small set of preferences as above, e.g. "person 3 is not similar to person 1, while person 2 is". Then we can find a weight vector $W = (w_1, \ldots, w_k)$ that stretches the attributes:

$$dist_W(A, B) = \sqrt{\sum_{i=1}^{k} w_i^2 (a_i - b_i)^2}$$

that gives rise to a clustering which (mostly) satisfies the expert opinion.

Of course scaling is the simplest way to induce a distance metric. More complex manipulation on non-Euclidean metric is possible. However for this paper, we shall restrict our attention to Euclidean space, and induce metric from linear transformation of the raw space.

One advantage of our approach to incorporate user preferences is that it is an wrapper method, which can be applied to any existing cluster algorithms without changing the algorithms.

# 2 Survey

## 2.1 Old fashioned Clustering

### 2.1.1 Statistical Clustering: Parametrical or Nonparametric

One of the classical references of clustering algorithm is Hartigan [8]. Chapter 5 of this book introduces the general method of model based clustering algorithm. This approach assumes that we know the number of clusters, say, $k$. The $d$-dimensional data points are assumed to be generated from a distribution with density function $f(x_1, \cdots, x_d)$ such that:

$f(x_1, \cdots, x_d) = \sum_{i=1}^{k} p_i f_i(x_1, \cdots, x_d|\vec{\theta}_i)$

where $p_i > 0$, $\sum_i p_i = 1$, and $f_i$ is the density function of the $i$th distribution, with parameter vector $\vec{\theta}_i$.

An iterative algorithm is proposed to find the values of $p_i$'s and $\vec{\theta}_i$'s. As we now know, it is a version of EM algorithm. (This book was published two years before Demptster et al. [5]). In particular, the author talks about the case where the $f_i$'s are assumed to be the density functions of multivariate normal distributions. To reduce the number of parameters to be estimated, the author also presents several ways to simplify the general algorithm. Basically, the author imposes more constraints on the density functions, e.g. they share the same covariance matrices, the covariance matrices are diagonal, etc.

Fraley and Raftery [7] presents a modern approach to the clustering problem based on parametric models. In particular, they tackle the problem of number of clusters. The classical approach assumes that we already know the number of clusters, which, in practice is often not the case. Another problem of the classical approach is that, although we could impose various constraints on the density functions to get different parametric models, in general, we do not know which model is the best one. To deal with these problems, Fraley and Raftery propose to run the EM algorithm on different number of clusters and different kind of parametric models. (To speed the rate of convergence, the input to the EM algorithm is preprocessed by the agglomerative hierarchical clustering method). Then, the BIC score function is used to compare the fitted models. They claim that this approach, although computationally expensive, gives better performance than the classical method.

The model based clustering algorithms assume that all the clusters have some regular shapes. For example, all the mixture of normal models based clustering algorithms assume that all the clusters should be either spherical or ellipsoidal. In practice, there are many cases where this assumption is not even approximately true, e.g., the problem of pattern recognition. In these cases, the nonparametric

approach to clustering is more appropriate.

Cuevas et al. [3] presents a clustering algorithm based on nonparametric density estimation. The basic idea of this approach is that, each cluster is assumed to be a connected component of the set $S = \{\vec{x} : f(\vec{x}) > c\}$, where $f$ is the density of the underlying distribution, and $c$ is some positive number. To find all the components in $S$, the author suggests the following algorithm: First, estimate the density $\hat{f}$ from the sample using some density estimation method, e.g., kernel density estimation. Choose a positive $c$ based on some criterion, e.g., $P(\{\vec{x} : f(\vec{x}) \leq c\}) = a$, where $a$ is some positive number between 0 and 1. (The larger the $a$ is, the weaker the cluster structure is.) Set a radius $\epsilon > 0$. (The authors give some suggestion regarding how to choose $\epsilon$.) Then include 2 points $\vec{x}_i$ and $\vec{x}_j$ in the same cluster if and only if $|\vec{x}_i - \vec{x}_j| < 2\epsilon$, $\hat{f}(\vec{x}_i) > c$, and $\hat{f}(\vec{x}_j) > c$. Bootstrap method can be used to estimate the variability of the result of the clustering algorithm.

### 2.1.2 Agglomerative and Divisive Clustering

For data points that are characterized by distances between them instead of coordinates, clustering is generally more difficult. The most intuitive and popular approach is hierarchical clustering: at each level a single point or group of points is joined with another (agglomerative methods), or a group of points is divided (divisive methods). Agglomerative methods are usually preferred since they require less computation. In [9], Hoffman and Buchmann applied the deterministic annealing technique to pairwise clustering. Their method uses simulated annealing to stochastically optimizing a cost function, which is derived according to maximum entropy inference. It is an influential work in pairwise clustering.

Tishby et al. recently introduced another method, which they call information bottleneck method [14] [13]. In their approach, the clustering problem is formalized as finding a short representation Y for one set of points X that preserves the maximum information about another set Z (Z can be a set of given categories). This yields a variational problem of maximizing the mutual information I(Y,Z) between Y and Z, subject to a constrain on the mutual information I(X,Y) between X and Y. In such a formulation, I(Y,Z) represents the information about Z encoded in Y, and I(X,Y) shows how compact the representation Y of X is. They implemented this idea as an iterative re-estimation method: starting by initialize Z as the same as X, then iteratively merge a pair of clusters in Z that minimizes the reduction of mutual information between Z and Y until stop criterion met. Although their method is computationally heavy, their idea provides a rich framework for discussing a variety of problems in learning.

## 2.2 Large Dataset Clustering

The BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) system, which is introduced in [15] by Zhang et al., dismisses EM algorithm because of the large amount of data accesses

required by EM. It incrementally and dynamically clusters incoming high dimensional data points to produce the best quality clustering with the available computing time and memory resources. By doing incremental clustering based on an in-memory, height-balanced and highly-occupied clustering feature tree, it minimizes the number of I/O operations and memory requirements. Each clustering decision is made without scanning all data points or all currently existing clusters, and is thus faster than global clustering methods. It's popular in the database community because of its ability to deal with very large data sets with relatively limited resources.

The problem of BIRCH is that it is unable to identify second-moment features of clusters, such as non-axis-aligned spread. In [12], Moore came up with an EM-based mixture model clustering algorithm using the multi-resolution kd-trees of [11] that greatly reduced the computation and I/O cost of traditional EM-based methods. This algorithm constructs an mrkd-tree top-down, identifying the bounding box of the current node, and splitting in the center of the widest dimension. By computing the minimum and maximum ownership that any point inside the node could have, many prunes can be made, and thus reduce the search space at every iteration of EM. With this saving on I/O operations and computations, we can use our good-old EM-based mixture model clustering to deal with larger data sets.

In [10] Pelleg and Moore extended the popular K-means with an estimation of the number of clusters. The resulting algorithm (named X-means) adds an "improve-structure" operation to the conventional K-means algorithm: in each iteration, after the K-means converge, find out if and where new centroids should appear. This new-centroids searching process is as follows: split every cluster centroids, do K-means in every cluster with the new centroids; then accept those new centroids that pass a Bayesian Information Criterion based test. The whole process, including the K-means step and the improve-structure step, is accelerated by operations similar to those used in [12]. This algorithm shows good performance in medium dimensionality (4 to 7) problems.

### 2.3 New Direction in Clustering: Put Human in the Loop

As we stated earlier, one major goal of clustering is for the domain experts to gain insight into the dataset. All clustering methods discussed above make assumptions, e.g. the distance function, the distribution, etc., which may not hold in practice. As a result, the single 'best' clustering may not truly reflect the nature of the dataset.

Metaclustering [1] is a new technique that addresses this problem. Instead of find the single 'best' clustering, metaclustering generates a large number of different but reasonable clusterings, and let the users (domain experts) pick a satisfactory one. To help users examine the vast variety of clusterings, metaclustering clusters clusterings according to some predefined similarity measure on clusters (hence the name) before presenting them to the users. As a result, similar clusterings are grouped together, and the users only need to examine the typical ones of each group. In this paper, multiple clusterings were

generated from different samples of the dataset by sampling with replacement (the so called bagging). They tried several different similarity measures between clusterings, the one based on hop distance seemed good. The basic clustering method was agglomerative clustering. They tried metaclustering on a synthetic dataset and the protein dataset with promising results. The main advantage of metaclustering is that users can actively choose the clustering they want with relatively small effort. The disadvantage is efficiency, since it has to generate a large number of different clustering, even though most of which may not be favored by the users and are never looked at. Unlike our proposed method here, the domain experts has no control over the process before everything has settled down.

Cohn, Caruana and McCallum proposed "semi-supervised clustering" [2] which has almost exactly the same basic idea as our proposed work: let the domain experts express their knowledge in the form of a few constraints on some data points, and the clustering algorithm takes into account these constraints to find a clustering that are both good in itself and in satisfying the constraints. Unfortunately their work was unpublished and we independently come up with the same idea later. Still, our proposed work has some important differences: First, they only implemented constraints of the form "point A and B should not be in the same cluster", while we plan to support more kinds of constraints (see sections below); Second, they worked on text document clustering problem where only the distance between documents is known, but we assume explicit vector representation of data points; In addition they assume an iterative interaction between the clustering algorithm and the user, while we assume all constraints are given a priori. And finally they modify the distance function between documents, while we weight attributes.

Demiriz, Bennett and Embrechts [4] also worked on similar problems. They assume the user can provide class labels to a few data points, which is equivalent to having a small training set. Then they perform clustering on the whole dataset, with the goal of finding a good clustering that are 'pure', i.e. datapoints within one cluster should have the same class label. The clustering can then be used to induce the classification on unlabeled data. They used genetic algorithms to find such clusterings.

Their work can be categorized as supervised learning with unlabeled data. While our work is unsupervised learning with 'weakly labeled' data. Their assumption that user can provide a few class labels is qualitatively different from our assumption on expert preferences. By class labels they implicitly assume the user knows how many classes there are, and which sample belongs to which class. This is a much stronger assumption. In contrast, we don't have the notion of classes. The user only need to specify whether two data points are similar, should be in the same cluster, or should not be in the same cluster. There is no need to assign class to data points. The relationship between classes and clusters is complex: a class can contain several clusters, or a cluster can have data points from several classes. We believe our scheme would be much easier for the user.

# 3 Technical Approach

To simplify the problem, we make the following assumptions:

- Data points are represented as a $k$ dimensional vector $A = (a_1, a_2, \ldots, a_k)$.
- The number of clusters is predetermined and fixed.
- We are given a working clustering algorithm (e.g. EM with mixture of Gaussian, or agglomerative clustering algorithm) which we will use as a black box.

Prior to clustering, the expert examines the dataset and gives a set of preferences $P = \{p_1, \ldots, p_m\}$. Each preference $p_i$ can have one of the following form:

1. Data points $A$ and $B$ shall be in the same cluster.
2. $A$ and $B$ shall be in different clusters.
3. $A$, $B$ are closer than $C$, $D$.

Our objective is to find a new Euclidean distance metric, or equivalently a $k \times k$ weight matrix $W = [w_{ij}]$ that linearly transforms data points: The weight will be applied to all data points to get $A' = WA$ so that the transformed dataset, when given to the black box clustering algorithm, results in a clustering that satisfies the maximal number of preferences.

We propose to use an iterative process and heuristic functions to find such a weight matrix. We start from an identity weight matrix $W^0$ and cluster the dataset. For each user preference that is violated by the clustering, we adjust the weight matrix heuristically. After adjusting the weight matrix for every violated preference, we apply it to the dataset and cluster the transformed dataset again. We hope the new weight leads to smaller number of violations. The weight update process is then repeated until we can't satisfy more preferences, or the iteration has reached a predefined number of times. Of course the user can dynamically add/remove/modify preference during the iteration. The procedure is shown in Figure 1.

We propose 4 different heuristics. The first 2 heuristics will adjust not the whole weight matrix, but only the diagonal elements of the weight matrix. That is, they only adjust the scales of the coordinates, but do not rotate the coordinate system. The latter 2 heuristics will adjust the whole weight matrix and thus allow the rotation of the coordinate system.

## 3.1 Heuristic 1

Heuristic 1 is tried as an improvement criterion in iterated clusterings.

The cost functions for individual preferences are as follows:

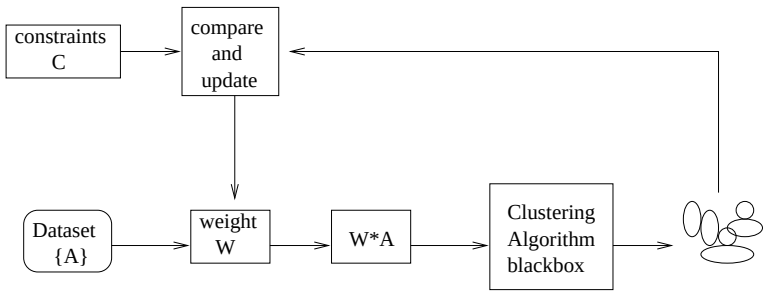


Figure 1: Flow chart of the proposed method

| Preference type | Cost function $Q_{p_i}(w_{11}, \cdots w_{kk})$ | Intuition |
|---|---|---|
| $A$ and $B$ not in the same cluster | $-dist_W(A, B)$ | Enlarge the distance |
| $A$ and $B$ in the same cluster | $dist_W(A, B)$ | Reduce the distance |
| $A$ and $B$ closer than $C$ and $D$ | $dist_W(A, B) - dist_W(C, D)$ | Enlarge the difference of distances |

where $p_i$ is the $i^{th}$ preference, $(w_{11}, \cdots, w_{kk})$ are the diagonal elements of the weight matrix, and $A, B, C$ and $D$ are data points.

The final cost function we try to minimize is the sum of the cost functions of all the preferences:

$Q(w_{11}, \cdots, w_{kk}) = \sum_{i=1}^{m}(Q_{p_i}(w_{11}, \cdots, w_{kk}))$

where $m$ is the number of preferences.

When the heuristic is used in a preprocessing step before any clustering attempts, the summations are performed over all the preferences provided by the expert. We try to minimize this cost function by modifying the diagonal elements of the weight matrix $W$ subject to a constraint that the sum of squared diagonal elements of $W$ is kept to 1. This can be done by solving a constrained optimization problem.

One problem with this preprocessing approach is that there are often not sufficient number of expert-provided preferences to justify such a greedy optimization. In such cases, we incorporate this heuristic in a series of clustering attempts: after each clustering attempt, slightly modify the weight matrix $W$ according to the cost function and renormalize it. The modification takes the following form:

$$w'_{ii} = w_{ii} - \lambda * \frac{\partial Q}{\partial w_{ii}}$$

$$w_{ii} = \frac{w'_{ii}}{||w'_{ii}||}$$

Here, the summations in computing the cost function $Q(W)$ can be taken over two possible sets of preferences:

1. The whole set that the expert provided, or

2. the preferences that are violated by current clustering result.

### 3.2 Heuristic 2

#### 3.2.1 How to force two points into the same cluster:

Without loss of generality, suppose that the user believes that the two data points $A$ and $B$ should belong to the same cluster, but were assigned to different clusters by the previous clustering procedure. We want to change the weight vector for the $k$ coordinates so that $A$ and $B$ will be put into the same cluster in the next clustering procedure.

The basic idea of our heuristic is putting less weight on the coordinate $i$ along which the two points $D_{1*}$ and $D_{2*}$ are farthest away from each other, or putting more weight on the coordinate $j$ along which $D_{1*}$ and $D_{2*}$ are closest to each other. One measurement of the closeness of two data points on a coordinate is the ratio of the distance between the two points along this coordinate to the weighted average within cluster standard deviation $\epsilon_i$ on that coordinate. [1]

Formally, we choose a coordinate $j$ such that:

$$j = \text{argmax}_{1 \le i \le k} \left\{ \left| \log \frac{(a_i - b_i)^2}{\epsilon_i^2} \right| \right\}$$

The weight $w_{jj}$ for the $j^{th}$ coordinate will be called the best weight to change (relative to the violation of the preference that $A$ and $B$ should be in the same cluster.) If the old weight for the $j^{th}$ coordinate is $w_{jj}$, then the new weight $w'_{jj}$ is:

$$w'_{jj} = \left[ 1 - \text{sgn} \left( \log \frac{(a_i - b_i)^2}{\epsilon_i^2} \right) \lambda \right] w_{jj}$$

where $\text{sgn}(x)$ is the sign function such that $\text{sgn}(x) = 1$ if $x > 0$, $\text{sgn}(x) = -1$ if $x < 0$, and $\lambda$ is a parameter used to adjust the step of the increment of the weights.

#### 3.2.2 How to force two points into different clusters

Suppose for now that $A$ and $B$ are believed to belong to different clusters, but were put into the same cluster by the clustering algorithm. By slightly modifying the above heuristic, we can get a heuristic to change the weight vector to force them into different clusters. We first find the coordinate $j$ in exactly

[1] Suppose the data set, which contains $n$ data points $D_1, \cdots, D_n$, is clustered into $s$ clusters, each of size $n_1, \cdots, n_s$. Then for coordinate $i$, $\epsilon_i$ is defined as:

$$\epsilon_i^2 = \sum_{j=1}^{s} \frac{n_j}{n} \frac{\sum_{l:\, D_l \text{ in cluster}_j} (d_{li} - \mu_{ij})^2}{n_j - 1}$$

where $\mu_{ij} = \sum_{l:\, D_l \text{ in cluster}_j} \frac{d_{li}}{n_j}$, and $d_{li}$ is the value of the $i^{th}$ coordinate for data point $D_l$.

the same way as before. As a matter of fact, the only modification we need to do is to modify the weight for the $j^{th}$ coordinate in the opposite way:

$$w'_{jj} = \left[1 + \text{sgn}\left(\log \frac{(a_i - b_i)^2}{\epsilon_i^2}\right)\lambda\right] w_{jj}$$

### 3.2.3 How to force a pair of points closer than another pair of points

Now suppose that we have 4 data points $A$, $B$, $D$, and $D$. The user believes that $A$ and $B$ should be closer to each other than $C$ and $D$, but the Euclidean distance between $A$ and $B$, given the current weight vector, is greater than the distance between $C$ and $D$. The heuristic of adjusting weight vector to satisfy the user's requirements is straightforward.

Note that $dist_W(C, D) - dist_W(A, B)$ can be expressed as a function of the diagonal elements of the weight matrix $\{w_{11}, \cdots, w_{kk}\}$. Compute the partial derivative of $dist_W(C, D) - dist_W(A, B)$ with respect to $\{w_{11}, \cdots, w_{kk}\}$. Pick up the coordinate $j$ where the partial derivative has the maximum absolute value:

$$j = \text{argmax}_{1 \le i \le k} \left\{ \left| \frac{\partial}{\partial w_{ii}} [dist_W(C, D) - dist_W(A, B)] \right| \right\}$$

Then decrease $w_{jj}$ if the partial derivative is positive, and increase $w_{jj}$ is the partial derivative is negative:

$w'_{jj} = w_{jj} * \left\{1 - \text{sgn}\left[\frac{\partial}{\partial w_i}[dist_W(C, D) - dist_W(A, B)]\right]\lambda\right\}$

### 3.2.4 Add Randomness

One concern with the above algorithm is that it is possible that we have two violations, and to solve one violation we need to increase the weight on a certain coordinate, while to solve another violation we need to decrease the weight on the same coordinate. We could mitigate this problem by incorporating a certain degree of randomness into our algorithm in the following way:

For each violation, we order the $k$ coordinates according to which one is best for change, which one is the second best, etc. Then we change $w_{ii}$ with a higher probability if the corresponding coordinate has a higher rank, and with a lower probability otherwise.

## 3.3 Heuristic 3

Heuristic 3 runs in sequential mode. That is, each violated preference is inspected in turn, and a slight change to the weight matrix is applied, as in the algorithm below:

```
W = Identity matrix
cluster(data, W)
```

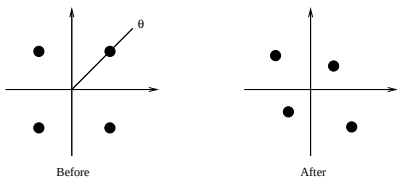


Figure 2: Shrink the space along direction $\theta$.

```
WHILE (preferences not well satisfied)
    FOR each violated preference p
        W <- adjust(W, p)
    END FOR
    cluster(data, W)
END WHILE
```

Note there's a small caveat: we really should re-cluster after each adjustment to $W$, but that's computationally too expensive.

There is only one adjustment action: to shrink the space along a certain direction. Figure 2 illustrates the effect of shrinking along direction $\theta$ in two dimensional space. The shrinkage can be achieved by first rotating the space so that the specified direction is on to, say, the first axis. Then we scale the first axis, and finally rotate the space back to the direction. These can be trivially implemented by matrix multiplication.

According to the type of preference $p$, there are three heuristics to pick a direction to shrink respectively, which we discuss below.

1. $p$: $A, B$ should not be in the same cluster

   When such a preference $p$ is violated, $A$ and $B$ are indeed being clustered together, as show in Figure 3. The intuition behind our heuristic for this problem is: we want to find a different cluster to move one of $A$ or $B$ to. Let $C_1, \cdots, C_k$ be the centroids of current clusters. W.L.G. assuming $A$ and $B$ are in cluster 1, and $d_W(A, C_1) > d_W(B, C_1)$. We will find a cluster centroid

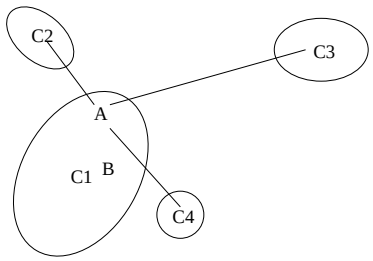


Figure 3: The preference "A and B should not be in the same cluster" is violated. In this example, we will heuristically shrink the space along direction $A - C_3$, since it is most orthogonal to $AB$.

$C_i, i \in \{2, \cdots, k\}$ to 'pull' $A$ towards. To do this, we shrink the space along the direction $A - C_i$ to decrease the distance between $A$ and $C_i$. Note the shrink of space will affect the distance between $A$ and $B$ as well, and we hope this distance remain as large as possible.

Formally, let $W_{AC_i}$ be the new weight matrix that shrinks the space along direction $A - C_i$ by some fixed coefficient $\eta \in (0, 1)$. Among $k - 1$ possible directions, we find the best one such that

$$W_{AC_*} = \arg\max_{i=2\cdots k} \frac{d_{W_{AC_i}}(A, B)}{d_{W_{AC_i}}(A, C_i)} - \frac{d_W(A, B)}{d_W(A, C_i)} \tag{1}$$

Shrinking along this best direction pulls $A$ closest to a different cluster $C_*$, while maintaining the distance between $A$ and $B$ relatively large. We thus hope in the next iteration $A$ and $B$ will be separated into different clusters. Note in this heuristic we ignore the closeness of different cluster centroids to $A$ to simplify computation.

2. $p : A, B$ should be in the same cluster

   When this preference is violated, $A$ and $B$ are in different clusters. Our heuristic is relatively simple here: we shrink the space along the direction of $A - B$ to bring $A$ and $B$ closer, so they might end up in the same cluster in next iteration.

3. $p : A, B$ should be closer than $C, D$

   When this preference is violated, the distance between $A$ and $B$ are larger than that between $C$ and $D$. We can use exactly the same heuristic as above: shrink along direction $A - B$ and

essentially ignore $C$ and $D$.

Once we find the best direction for each case above, we will shrink along the direction by amount $\eta_p^\alpha$, where $\eta \in (0,1)$ is a fixed step size, and $\alpha_p \in (0,1]$ is the user confidence on preference $p$. The motivation is to be conservative if the user is not very sure.

### 3.4 Heuristic 4

Heuristic 4 is a simplification to heuristic 3. It is exactly the same as heuristic 3 except that when the preference "$p$: $A, B$ should not be in the same cluster" is violated, we simply *stretch* along the direction $AB$.

## 4 Experiments

### 4.1 Datasets

The datasets that we use are:

- Synthetic weight, height data of female and male

  The goal is to cluster samples into females and males respectively. This dataset illustrates the importance of correct scaling of attributes. This dataset contains 96 samples, 48 females and 48 males. Each sample has 2 dimensions: the weight and the height. In the dataset, the height is in meter and weight in kg. When clustering with the raw Euclidean distance, the resulting clusters don't correspond to gender difference (weight over dominates). If height is scaled, e.g. to cm, the result will correspond to gender, see figure 4. The heuristics should be able to find the appropriate scaling.

- Digits handwriting dataset

  This dataset is extracted from an original data set of digitized handwriting trajectories of digits 0 to 9. The new data set contains 75 data points and 16 features for 5 digits. Without making any modification to this data set, the kmeans algorithm can cluster it perfectly into 5 clusters such that each cluster contains data points for exactly one digit. Our use of this data set is to assign random weight to each feature to generate a modified data set. Applying kmeans algorithm to this modified data directly will yield a bad clustering result, i.e., each cluster contains data points for several different digits. Then we can generate some user preferences based on our information about which data point is about which digit, and apply our weight adjusting algorithm to the modified data set to see whether these preferences can guide us to cluster the data sets perfectly.

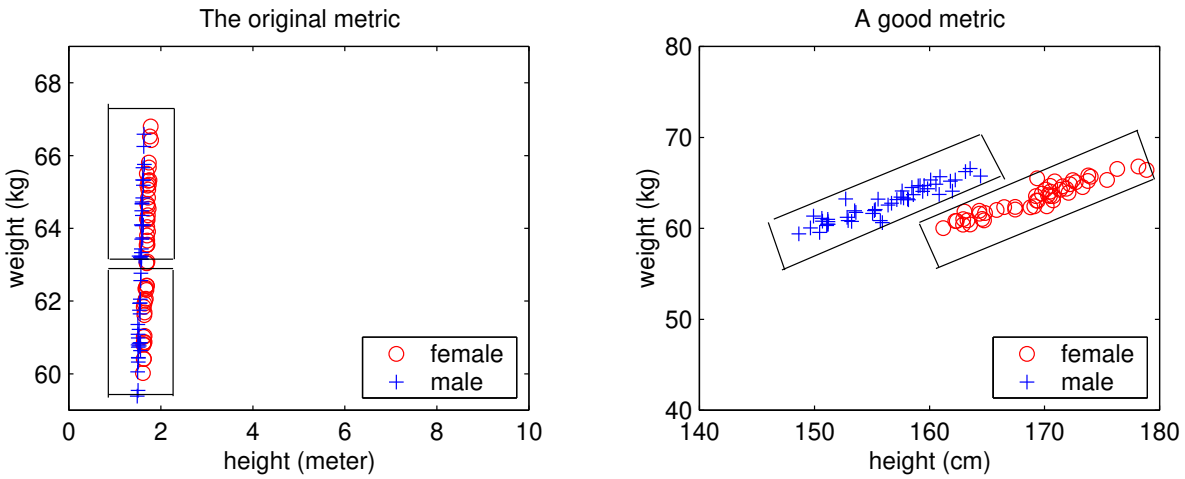


Figure 4: The importance of metric to clustering for the weight-height data

- Usenet text documents

  400 Internet discussion group text documents from 4 Usenet discussion groups: real car, car simulator, real aviation, and aviation simulator. The dataset can be divided in two ways:

  1. Aviation vs. automobile
  2. Simulation vs. real

  which can be specified as user preference. We used tf.idf vector as feature representation for documents. We performed aggressive feature selection by document frequencies. Each vector has 437 dimensions. We want to show that by giving different preferences, we can guide the clustering towards different division.

### 4.2 Generating Preferences

Since we know the class label of the samples, we can generate a preference randomly by:

- randomly pick two samples $A, B$ with the same class label, create a preference "$A, B$ should be in the same cluster".
- randomly pick two samples $A, B$ with different class labels, create a preference "$A, B$ should not be in the same cluster".
- randomly pick two samples $A, B$ with the same class label, and $C, D$ with different class labels, and create a preference "$dist(A, B) < dist(C, D)$"

Although we use class labels in generating the preferences, it is different than giving out a few class labels, thus different than supervised learning with unlabeled data.

For this experiment, we generate a set of random preferences *a priori*, and use this set of preferences through out the iterations. This scheme corresponds to the situation where a domain expert examines the data set before doing clustering, and gives some expert opinions. Note some preferences may be already satisfied even with the original distance metric. And as more and more preferences are satisfied during iterations, we are left with less and less violated preferences to guide the metric induction. An alternative scheme is to dynamically generate preferences that are violated after each iteration of clustering. This corresponds to the domain expert looking at clustering results at each iteration, and picking out a few 'mistakes' made by the clustering algorithm. Both schemes are reasonable and we may combine them in the future. However in the paper we didn't pursue the alternative scheme.

### 4.3 Clustering 'black boxes'

We use two very different clustering programs as the 'black boxes', to show that our heuristics can be applied to existing clustering algorithms without change. The two programs are K-means clustering of Andrew Moore's group which employs the EM algorithm, and a hierarchical agglomerative clustering program by Rich Caruana's group.

### 4.4 Results

For a given set of preferences, we use our heuristics to iteratively adjust the weight matrix (and hence the distance metric), and stop when all the preferences have been satisfied, or when the iteration exceeds a predetermined number (e.g. 500). Then we measure the *purity* of the resulting clusters. Purity is defined as the sum of the majority class samples in all clusters, divided by the total number of samples. A purity of 1 means the clustering corresponds exactly to the underlying class labels. The worst purity is 0.5.

Figure 5 shows the purity of the final clustering on the weight-height data, after maximally satisfied the preferences. As we can see, the more preferences a user gives the system, the better purity the final clusters have. The left most point (zero preference) corresponds to clustering with the original Euclidean distance metric. Evidently all the heuristics helped.

Figure 6 shows the purity on the digit data set. Again, all heuristics helped. And the number of preferences is positively correlated to purity.

The Usenet text data set, as we have seen, has two possible divisions. First, we generate preferences such that the division is 'simulation' vs. 'real', e.g. a document about auto simulator is considered to be in the same cluster as a document about aviation simulator. The two underlying classes are 'simulation' and 'real' too. Table 1 shows some preliminary purity results (a bug in the clustering code prevented us from getting all the results. We are working on it).

Second, we generate preferences along the 'car' vs. 'aviation' division, so are the two underlying

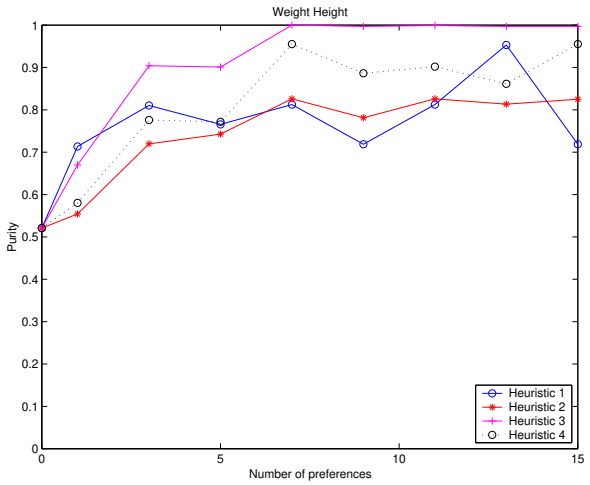


Figure 5: Purity of the weight-height dataset, with different number of preferences

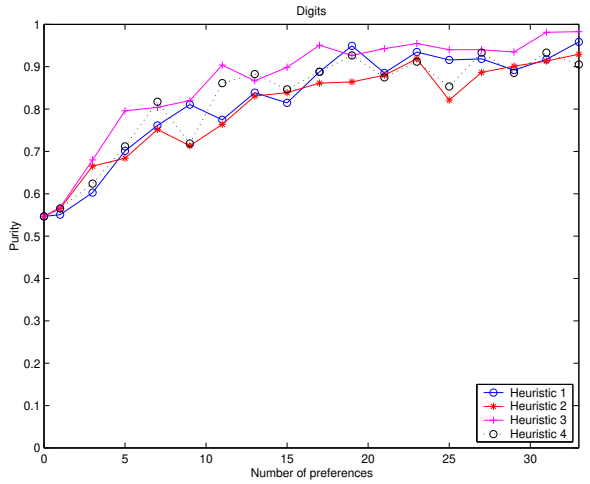


Figure 6: Purity of the digits dataset, with different number of preferences

| # of preferences | Heuristic 1 | Heuristic 4 |
|---|---|---|
| 0 | 0.755 | 0.755 |
| 10 | 0.72 | |
| 20 | | 0.685 |
| 40 | | 0.535 |
| 60 | | 0.82 |

Table 1: Purity results for the Usenet sample: 1

classes. Table 2 shows the results.

| # of preferences | Heuristic 1 | Heuristic 4 |
|---|---|---|
| 0 | 0.545 | 0.545 |
| 10 | | 0.605 |
| 20 | 0.56 | 0.6325 |
| 40 | | 0.7475 |
| 60 | 0.62 | 0.58 |
| 80 | 0.62 | 0.7375 |

Table 2: Purity results for the Usenet sample: 2

Interestingly, although in most cases we satisfied all preferences, the purity was not very high. There seems to be some form of overfitting (remember we have more than 400 features).

## 5 Discussion

In the experiments, we tested the perfectness of the clustering results with the changed weight matrix or vector. This is measured by "purity", which is the sum of number of samples with the majority class label within each class, divided by the total number of samples. Ideally purity=1. But during clustering, this purity is unknown to the weight adjusting algorithms. Based on the experiment results, we make the following observations:

1. One major idea of this project is to test the feasibility of using some expert-defined preferences to guide us in optimizing the "purity" of the clustering results: the weight adjusting algorithms try to optimize the satisfaction of the set of preferences, which is measured by "happiness" – the sum of satisfied preferences weighted by the expert's confidence in each preference. The happiness function can be quite different from the purity function in the weight matrices space: the local monotonicity can be different from each other; even the global maximum can be different – neither does 100% satisfaction of the preferences mean purity 1, nor do weight matrices that achieve purity 1 necessarily satisfy all the "closer" type preferences. This problem is especially severe for high-dimension tasks. In the experiments, we often observed that the weight adjusting algorithms keep going even when a purity of 1 has been achieved.

2. In general, if the heuristic allows the rotation of the coordinate system, i.e., the heuristic could update the whole weight matrix, not only the diagonal of the weight matrix, we can get better clustering result.

3. In general, the performance of our algorithms will increase as the number of preferences increase. On the other hand, after the number of preferences has reached a certain level, the gain of the performance by increasing the number of preference is negligible.

4. Our heuristics work well for low and medium dimensionality data sets. It performs reasonably well when the dimension is 16 (digits data). The experiment results with the Usenet data (400+ dimensions) are not satisfactory, and some overfitting (all preferences satisfied, but purity is low) seemed to had happened. A bug in the kmeans clustering program prevents us from getting a clear picture of the situation at this moment. We are going to continue working on this.

We can gain some insights into our metric inducing heuristics by considering the *inductive bias* introduced by the metric change. To illustrate the idea, let's focus on a preference which is the easiest to understand: "$d(A, B) < d(C, D)$". Consider a hypothetical algorithm which, in order to satisfy this preference, performs the following operations:

1. The algorithm computes all pairwise distances $d(D_i, D_j)$ between samples.

2. The algorithm explicitly shortens $d(A, B)$ so it is smaller than $d(C, D)$. All other distances remain unchanged.

3. The algorithm then performs multi-dimensional scaling (MDS) on all distances, including the shortened $d(A, B)$.

After MDS, we get a new set of coordinates for all samples. Assuming MDS is perfect, then the set of new coordinates would have satisfied the preference, with everything else (in terms of distance metric) unchanged. This is of course not what a user wants, since the clustering is probably going to be the same as before. The user expects the preference to induce a wider change. Such a hypothetical algorithm is analogous to an unbiased rote learner in machine learning. To make our heuristics useful, we have to introduce inductive biases. In heuristic 1 and heuristic 2, the inductive bias is expressed by scaling coordinate-wise as a whole. In heuristic 3 and heuristic 4, the inductive bias is by scaling along arbitrary directions.

We are going to further our research by refining the heuristics, and experimenting on more datasets, including the Sloan astrophysics dataset.

## 6 Summary

We proposed a set of heuristics to incorporate expert preferences into the process of clustering. They have the potential to allow experts to specify domain knowledge with little effort and get satisfactory

clustering results. Experiment results are promising. We believe it will eventually be a valuable extension to standard clustering algorithms, and domain experts will want to keep it in their collection of datamining tools.

## References


[1] Rich Caruana, Pedro Artigas, Anna Goldenberg, and Anton Likhodedov. Meta clustering. submitted for publication, ICML2001.

[2] David Cohn, Rich Caruana, and Andrew McCallum. Semi-supervised clustering with user feedback, 2000. draft.

[3] A. Cuevas, M. Febrero, and F. Fraiman. Estimating the number of clusters. *The Canadian Journal of Statistics*, 2000.

[4] A. Demiriz, K. Bennett, and M. Embrechts. Semi-supervised clustering using genetic algorithms. In *ANNIE*, 1999.

[5] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977.

[6] Richard Duda, Peter Hart, and David Stork. *Pattern Classification, Second Edition.* Wiley, John & Sons, Incorporated, 2000.

[7] C. Fraley and A. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *Computer Journal*, 41:578–588, 1998.

[8] J. Hartigan. *Clustering Algorithms.* John Wiley & Sons, New York, 1975.

[9] T. Hoffmann and J. Buchmann. Pairwise data clustering by deterministic annealing. *IEEE Trans. on PAMI*, 19(1):1–14, 1997.

[10] A. Moore and D. Pelleg. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the International Conference on Machine Learning*, 2000.

[11] A. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *Proceedings of the 1997 International Machine Learning Conference*, 1997.

[12] Andrew Moore. Very fast mixture-model-based clustering using multiresolution kd-trees. In *Advances in Neural Information Processing Systems*, pages 543–549, 1999.

[13] N. Slonim and N. Tishby. Agglomerative information bottleneck. In *Proceedings of Neural Information Processing Systems*, 1999.

[14] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

[15] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large database. In *Proceedings of ACM SIGMOD*, pages 103–114, 1995.