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

April 10, 2001

Abstract

Traditional clustering algorithms cannot incorporate domain expert's preferences into the process of clustering. 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 an algorithm which takes into account the expert constraints and guides the clustering towards them by inducing an appropriate distance metric, so that the final clustering maximally satisfies the expert preferences while is still reasonably good in traditional measures.

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 between a few data points in the dataset. Note "a few" is the key word: experts are busy, or the dataset is too large to be read by human. The expert may say the following:

- "data point A looks like it should be in the same cluster as datapoint 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 metric on the same dataset, with the same clustering algorithm may produce different clustering result [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 raw 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:

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

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 ith 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 ϵ.) 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 multi-dimensional metric 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 multiresolution 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 median 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 semi-supervised clustering problems. Their work is more close to a classification with unlabeled data setting. 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. However the 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. Moreover, the relationship between classes and clusters could be 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 $A = (a_1, a_2, \ldots, a_k)$.

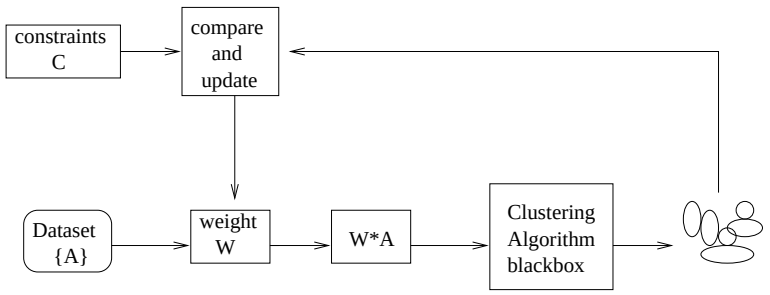

Figure 1: Flow chart of the proposed method

- 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 constraint 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 $W = (w_1, \ldots, w_k)$ 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 constraints.

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 constraint, 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.

At this stage each of us proposed competing heuristics, as detailed below. We will let experiments decide which heuristics will prevail.

3.1 Jie's heuristic

I propose a heuristic that will be tried both in a preprocessing step before any clustering and as an improvement criterion in iterated clusterings.

The cost function for this heuristic is as follows:

$Q(w) = -\sum(d_w(p_i, p_j)) + \sum(d_w(p_k, p_l)) - \sum(d_w(p_m, p_n) - d_w(p_s, p_t))$

where $c(p_i) \neq c(p_j)$, $c(p_k) = c(p_l)$, and $d(p_m, p_n) > d(p_s, p_t)$

The three components in this cost function deal respectively with the three types of constraints the expert provide:

1. p_i and p_j should not be in the same cluster;
2. p_k and p_l should be in the same cluster;
3. p_s, p_t should be closer than p_m, p_n.

When the heuristic is used in a preprocessing step before any clustering attempts, the summations are performed over all the constraints provided by the expert. We will try to minimize this cost function by modifying the weight vector w subject to a constraint that the magnitude of w is kept to 1. This can be done by solving a constrained optimization problem.

One possible problem with this preprocessing approach is that there may not be sufficient number of expert-provided constraints to justify such a greedy optimization. In case of this, we will also try to incorporate this heuristic in a series of clustering attempts: after every clustering attempt, slightly modify the weight vector w according to the cost function and renormalize it. The modification takes the following form:

$w = w - \lambda \frac{\partial Q(w)}{\partial w}$

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

1. The whole set that the expert provided, or
2. the constraints that are violated by current clustering result.

Both choices of constraint sets will be tried in our experiments.

3.2 Tianjiao's Heuristics

Suppose we have a k-dimensional data set of size n represented by an $n \times k$ matrix $D = \{d_{ij}\}$. That is, each row of D, call it D_{i*}, is a data point, and each column of D, call it D_{*j}, represents a coordinate. We also have a set of conditions given by the user. Each condition either specifies whether two data

points should be in the same cluster, or whether a pair of data points are closer to each other than another pair of data points. The user also gives numeric values between 0 and 1 to specify his/her degrees of confidence about these conditions.

Let $W = \{w_1, \cdots, w_k\}$ be a non-negative vector represents the relative weights of the k coordinates. The purpose of our algorithm is to search for a weight vector such that, by multiplying each column of D with the corresponding weight, then applying a clustering algorithm to the modified data, we should get a clustering result that satisfies as many conditions given by the user as possible.

3.2.1 Our algorithm

Our algorithm will be an iterative algorithm. In each iteration, we will first multiply each column of the sample with the weight given by the current weight vector, and run a clustering algorithm to the modified data. Then for each condition given by the user, we will check whether it is satisfied by the clustering result. If a condition is satisfied, we will do nothing. Otherwise, with a probability equal to the user's confidence about this condition, we will choose to change the weight vector in a way specified in the following sections.

We will to stop if all the conditions given by the user are satisfied, or if we have done N iterations, where N is some fixed constant, say, 100.

3.2.2 How to force two points into the same cluster:

Without loss of generality, suppose that the user believes that the two data points D_{1*} and D_{2*} 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 D_{1*} and D_{2*}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 of the two points along this coordinate to the weighted average within cluster standard deviation ϵ_i on that coordinate. [1]

Formally, we will choose a coordinate j such that:

[1] Suppose D is clustered into m clusters, each of size $m_1, \cdots, m_m$. Then for coordinate i, ϵ_i is defined as:

$$\epsilon_i^2 = \sum_{j=1}^{m} \frac{m_j}{m} \frac{\sum_{l:\, D_{l*} \text{ in cluster}_j} (d_{li} - \mu_{ij})^2}{m_j - 1}$$

where $\mu_{ij} = \sum_{l:\, D_{l*} \text{ in cluster}_j} \frac{d_{li}}{m_j}$.

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

The weight w_j for the j^{th} coordinate will be called the best weight to change (relative to the violation of the condition that D_{1*} and D_{2*} should be in the same cluster.) If the old weight for the j^{th} is w_j, then the new weight w_j' for the coordinate j is:

$$w_j' = \left[1 - \text{sgn} \left(\log \frac{(d_{1i} - d_{2i})^2}{\epsilon_i^2} \right) \lambda \right] w_j$$

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 λ is a parameter used to adjust the step of the increment of the weights.

3.2.3 How to force two points into different clusters

Suppose for now that D_{1*} and D_{2*} 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 will first find the coordinate j in the exactly 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:

$$w_j' = \left[1 + \text{sgn} \left(\log \frac{(d_{1i} - d_{2i})^2}{\epsilon_i^2} \right) \lambda \right] w_j$$

3.2.4 How to force a pair of points closer than another pair of points

Now suppose that we have 4 data points D_{1*}, D_{2*}, D_{3*}, and D_{4*}. The user believes that D_{1*} and D_{2*} should be closer to each other than D_{3*} and D_{4*}, but the Euclidean distance between D_{1*} and D_{2*}, given the current weight vector, is greater than the distance between D_{3*} and D_{4*}. The heuristic of adjusting weight vector to satisfy the user's requirements is straightforward.

Let $d(D_{i*}, D_{j*})$ be the Euclidean distance between D_{i*} and D_{j*}. Then $d(D_{3*}, D_{4*}) - d(D_{1*}, D_{2*})$ can be expressed as a function of the weight vector $W = \{w_1, \cdots, w_k\}$. Compute the partial derivative of $d(D_{3*}, D_{4*}) - d(D_{1*}, D_{2*})$ with respect to $w_1, \cdots, w_k$. Pick up the coordinate j where the partial derivative has the maximum absolute value:

$$j = \text{argmax}_{1 \le i \le k} \left\{ \left| \frac{\partial}{\partial w_i} [d(D_{3*}, D_{4*}) - d(D_{1*}, D_{2*})] \right| \right\}$$

Then decrease w_j if the partial derivative is positive, and increase w_j is the partial derivative is negative:

$w_j' = w_j * \left\{ 1 - \text{sgn} \left[\frac{\partial}{\partial w_i} [d(D_{3*}, D_{4*}) - d(D_{1*}, D_{2*})] \right] \lambda \right\}$

3.2.5 Room for improvement

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 find s best weights $w_{j1}, \cdots, w_{js}$ to change, where $1 \leq s \leq k$, in a similar way as we find the best weight to change. Then we change w_{ji} with probability p_i, where $1 \leq i \leq s$ and $\sum_{i=1}^{s} p_i = 1$. Suppose w_{ji} is the i^{th} best weight to change, we could assign p_i in the following way:

$p_i = \dfrac{2^{-i}}{1 - 2^{-s}}$

3.3 Jerry's heuristic

I propose a heuristic that 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)
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 θ in two dimensional space.

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 $C_i, i \in \{2, \cdots, k\}$ to 'pull' A towards. To do this, we shrink the space along the direction $A - C_i$ to

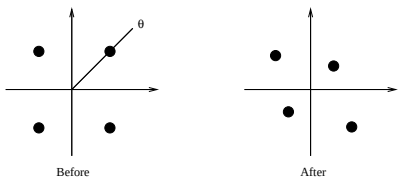

Figure 2: Shrink the space along direction θ.

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 have two choices here:

   (a) We can use exactly the same heuristic as above: shrink along direction $A - B$ and essentially ignore C and D.

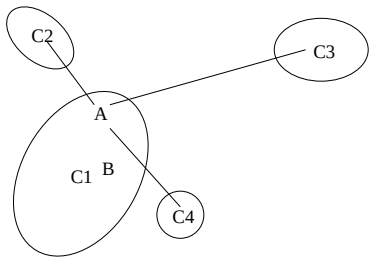

Figure 3: The preference "A and B should not be in the same cluster" is violated. In this example, we will heuristically shrink the space along direction $A - C_3$, since it is most orthogonal to AB.

(b) Find the best direction that maximizes $\frac{d_{W^*}(C,D)}{d_{W^*}(A,B)}$

Once we find the best direction for each case above, we will shrink along the direction by amount $\eta * \alpha_p$, where $\eta \in (0, 1)$ is a fixed step size, and $\alpha_p \in (0, 1]$ is the user confidence on preference p.

4 Experiments

4.1 Datasets

The datasets that we will use are:

- Synthetic weight, height data of female and male

  The goal is to cluster females and males respectively. This dataset illustrates the importance of correct scaling of attributes. 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. The heuristics should be able to find the appropriate scaling.

- Digits handwriting dataset

  This dataset contains handwriting trajectories of digits 0 to 9. This is a good dataset to illustrate the difference between classification and clustering, and the limit of the metric method. For

example, the same class '5' contains two clusters, depending on the order of the two strokes. And our proposed method may not be able to satisfy a user preference to put these two kinds of '5' into the same cluster.

- Usenet text documents

  400 Internet discussion group text documents. The dataset can be divided in two ways:

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

  which can be specified as user preference.

- Sloan astrophysics dataset

5 Discussion

We can gain some insight into our metric inducing method by considering the *inductive bias* introduced by the metric. More on this in the final paper.

6 Updated Work Plan

- Implement individual heuristics, test them on different datasets.
- Work with Bob Nichol for the expert preferences and interpretation of the Sloan data clustering. (joint work)
- Compare different heuristics (joint work)
- Project write up (joint work)

Please refer to Appendix A and B for the work we have done so far, and the old work plan.

7 Summary

We proposed a method to incorporate expert preferences into the process of clustering. The proposed research, if successful, will allow experts to specify domain knowledge with little effort and get satisfactory clustering results. We believe it will 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.

Appendix A: Past Work

- Jie
  - Processed the Usenet discussion group text dataset, turned the texts into vectors.
  - Acquired the Kmeans (Xmeans) clustering software and tested it.
  - Built programming API to the Kmeans clustering algorithm.
  - Proposed a set of heuristics for weight vector optimization.
- Tianjiao
  - Acquired two data sets and applied various clustering algorithms to them.
  - Proposed a set of heuristics for weight adjusting.
  - Implemented my heuristics.
- Xiaojin
  - created a synthetic weight/height dataset.
  - acquired the Sloan astrophysics dataset and performed elementary check on the data.
  - acquired the Usenet discussion group text dataset.
  - built programming API to the agglomerative clustering algorithm.
  - proposed a set of heuristics for metric induction.
  - wrote software framework for everybody to carry on the implementation.

Appendix B: Old plan

Our project will be a joint work in the genuine sense. We have spent, and will spend, a lot of time to discuss about ideas coming from various areas familiar to one or more of us, examine the relevance of these ideas to our project, and explore how these ideas could be incorporated into our approach. It is not always possible, in this work plan, to state explicitly which one of us will do which part of work. But we will try to specify how to divide some work among us when it makes sense.

Our project will be implemented in three stages:

- First stage: Feb. 1 — Feb. 15 Search for the literatures relative to our goal. Discuss possible approaches. Write the proposal.

  Jerry Zhu: Search for the literature about use of user input in clustering. Write the survey for this part of literature.

  Tianjiao Chu: Search for the statistical approaches to clustering algorithm. Write the survey for this part of literature.

  Jie Zhu: Search for the literature in large database clustering. Write the survey for this part of literature.

- Second stage: Feb. 19 — April 3 Implement the algorithms. Write the progress report.

  Jie Zhu: Collect data, implement the synthetic data generation algorithm.

  Tianjiao Chu: Implement the clustering algorithm.

  Jerry Zhu: Implement the algorithm to evaluate the clustering results.

- Third stage: April 4 — May 3 Test the algorithm both with synthetic data and real data. Evaluate the results. Discuss their implications and possible future works. Write the final report.

  Jie Zhu: Generate the data.

  Jerry Zhu: Create the user inputs.

  Tianjiao Chu: Run the algorithm.