Chapter 9

Data Clustering

In this chapter, we examine the problem of data clustering. Consider the two data sets in Figure 9.1. In the first graph, we see data that have no class labels, but there seems to be a natural separation of the data into three clumps. A natural clustering algorithm would produce three clusters, and this would be an unsupervised task. On the other hand, the data may include class labels, as seen in the second graph (Class 1 is triangles, Class 2 is asterisks), and in this supervised learning task, we would want to find a function that would tell us which class is assigned to each data point.

In this chapter, we review the main clustering algorithms currently in use. We will see that they share many common characteristics, including the assumption that the data set under study can be accurately clustered using spatial information alone. This leads us to consider a clustering algorithm designed specifically for attracting sets of dynamical systems, and culminates in the development of a space-time clustering algorithm.

9.1 Background

We will first look at the unsupervised clustering task. In this case, the input to the algorithm is a data set and the desired output is the number of clusters used, and the membership function that maps the data to its corresponding cluster index.

Definition: Unsupervised Clustering In the unsupervised task, we are given a data set, X, whose elements are vectors $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$. We want a membership function which has a domain in $\mathbb{R}^n$ and will output the cluster index (or label). Defining this function as m, we have:

$$m(\boldsymbol{x}^{(i)}) = g_i$$

where g_i is the integer for the class for the data point, typically one of the integers from 1 to k, where k may be specified or perhaps output from the algorithm.

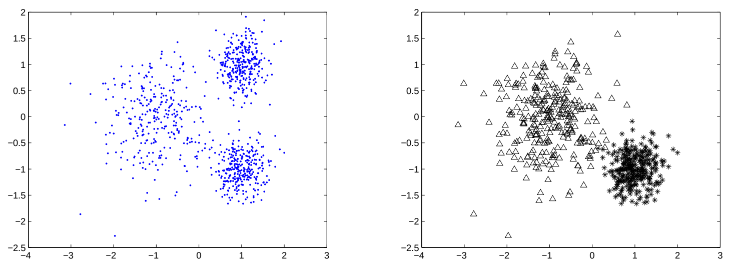

Figure 9.1: In the first graph, we see data that have no class labels, but there seems to be a natural separation of the data into three clumps. An ideal clustering algorithm would produce the three clusters, and this would be an unsupervised task. On the other hand, the data may include class labels, as seen in the second graph (Class 1 is triangles, Class 2 is asterisks), and in this supervised learning task, we would want to find a function that would tell us which class is assigned to each data point.

One of the things that makes this problem so interesting is that it is very ill-posed- there are an infinite number of ways of building a membership function. For two extreme examples, consider these two membership functions:

$$m_1(\boldsymbol{x}^{(i)}) = 1$$

where i is the index for the data points. Another function that is about as useful as m_1 is the following:

$$m_2(\boldsymbol{x}^{(i)}) = i$$

In the first case, we have only one class, and in the second case, we have as many classes as there are data points! In order to get a useful algorithm, we will need to try to define some kind of error function that we can then minimize. The easiest way to do this is through *Voronoi Cells*:

Definition: Let $\left\{\boldsymbol{c}^{(i)}\right\}_{i=1}^{k}$ be points in $\mathbb{R}^n$. These points form k Voronoi Cells, where the j^{th} cell is defined as the set of points that are closer to cell j than any other cluster:

$$V_j = \left\{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x} - \boldsymbol{c}^{(j)}\| \leq \|\boldsymbol{x} - \boldsymbol{c}^{(i)}\|,\ i = 1, 2, \ldots, k\right\}$$

The points $\left\{\boldsymbol{c}^{(i)}\right\}_{i=1}^{k}$ are called *cluster centers*. In the uncommon occurrence that a point $\boldsymbol{x}$ lies on the border between cells, it is customary to include it in the cell whose index is smaller (although one would fashion the decision on the problem at hand). The reader might note that a Voronoi cell has a piecewise linear border.

Examples in "Nature"

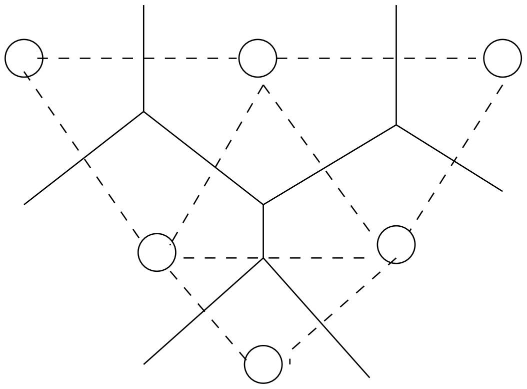

Figure 9.2: Sample Voronoi diagram sketched by hand. First, draw a line between neighboring centers (dotted in the figure). These are guides to drawing the *actual* borders, shown in solid black. These are perpendicular bisectors of the dotted lines.

1. In [**?**], Voronoi cells are used to define the "area potentially available around a tree". That is, each tree in a stand represents the center of the cell.
2. Using a map of the campus with the emergency telephone boxes marked out and used as the centers, we could always tell where the closest phone is located.

We can draw a Voronoi diagram by hand: Between neighboring cells, draw a line (that will be erased at the end), then draw the perpendicular bisectors for each line drawn. See Figure 9.2. This can get complicated fairly quickly, so we will be using Matlab to produce the plots. The algorithms that do these plots are very interesting (see, for example [**?**]) but will be beyond the scope of our text.

Matlab Example: Matlab has the the plotting algorithm built-in. For example, Figure 9.3 shows the output of the following (yours will be slightly different due to using random cluster centers):

```
X=randn(10,2);
voronoi(X(:,1),X(:,2));
```

We can also have Matlab return the vertices of the Voronoi cells to plot them manually:

```
[vx,vy]=voronoi(X(:,1),X(:,2));
plot(vx,vy,'k-',X(:,1),X(:,2),'r*');
```

In the algorithms that we work with, it will be convenient to have a function that will identify those points within a given cluster; the characteristic function

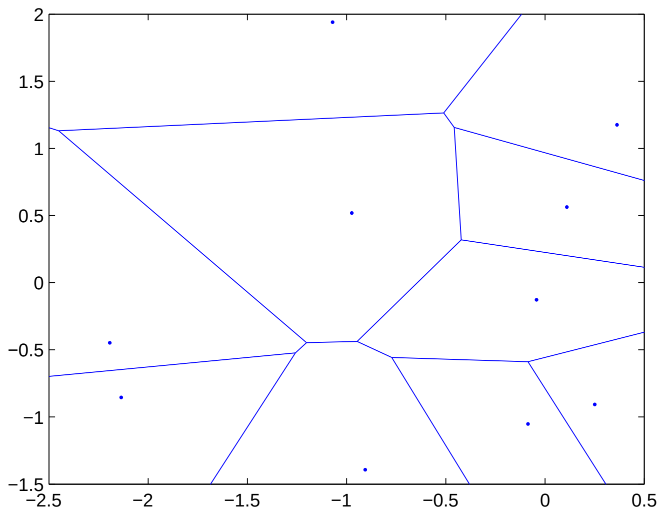

Figure 9.3: The output of our Matlab example, running the `voronoi` algorithm on 10 random points in $\mathbb{R}^2$.

is typical for that purpose, and is defined as 0 or 1, depending on whether or not the data point belongs to the cluster:

$$\chi_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } m(\boldsymbol{x}) = i \\ 0 & \text{otherwise} \end{cases}$$

One general way to measure the goodness of a clustering algorithm is to use a measure called the **distortion error** for a given cluster i, and the total distortion error[1]:

$$E_i = \frac{1}{N_i} \sum_{k=1}^{N} \|\boldsymbol{x}^{(k)} - \boldsymbol{c}^{(i)}\|^2 \, \chi_i(\boldsymbol{x}^{(k)}) \qquad E_{\text{total}} = \sum_{k=1}^{p} E_k$$

where N_i is the number of points in cluster i, and N is the total number of points overall. You might notice that the values being summed are only non-zero for those data points $\boldsymbol{x}$ that are in cluster i. The total error is simply the sum of the individual "errors" or distortions.

[1]Some authors do not square the norms, but it is more convenient for the theory, and does not change the end results.

9.2 The LBG Algorithm

The LBG (Linde-Buzo-Gray, see [22]) algorithm[2] is perhaps the fastest of this chapter, and the easiest to visualize. The membership function for the LBG algorithm is defined so that we are in cluster i if we are in the Voronoi cell defined by the i^{th} cluster center. That is,

$$m(\boldsymbol{x}) = i \text{ iff } \|\boldsymbol{x} - \boldsymbol{c}^{(i)}\| < \|\boldsymbol{x} - \boldsymbol{c}^{(j)}\| \quad i \neq j, \ \ j = 1 : p \tag{9.1}$$

In the rare circumstance that we have a tie, we'll choose to put the point with the center whose index is smallest (but this is ad-hoc).

The Algorithm

Let X be a matrix of p data points in $\mathbb{R}^n$, and let C denote a matrix of k centers in $\mathbb{R}^n$. At each pass of the algorithm, the membership function requires us to take $p \times k$ distance calculations, followed by p sorts (each point has to find its own cluster index), and this is where the algorithm takes most of the computational time. To do the distance calculations, we will write a function named `edm` (for Euclidean Distance Matrix). See the exercises in this section for the Matlab code.

The algorithm begins by choosing the centers to be data points (randomly selected), and then we iteratively update the centers that will minimize our error, E_{total}:

- Sort the data into k sets by using the membership function in Equation 9.1. We will use the EDM to do this.
- Re-set center $\boldsymbol{c}_i$ as the centroid of the data in the i^{th} set.
- Compute the distortion error.
- Repeat until the distortion error no longer decreases (either slows to nothing or starts increasing).

A Toy Example:

Here we use the LBG algorithm three times on a small data set, so you can have a test set for the code that you write.

Let the data set $X^{5\times 2}$ be the matrix whose rows are given by the data points in the plane:

$$\left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\}$$

[2]The LBG algorithm is similar to a method known as *k-means* clustering. We will define the $k-$means clustering to be the on-line version of the LBG.

You might try drawing these on a plane. You may see a clustering of three points to the right, and a clustering of two points to the left.

We will use two cluster centers, and initialize them randomly from the data. In this case, the rows of C are:

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The matrix matrix $D^{5\times 2}$ shows the result of using `edm` to compute the distances between our 5 data points and 2 cluster centers:

$$D = \begin{bmatrix} \sqrt{2} & 1 \\ 0 & 1 \\ \sqrt{2} & \sqrt{5} \\ 1 & 0 \\ \sqrt{5} & \sqrt{8} \end{bmatrix}, \qquad M = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

You can see that M gives the membership index (the value of the smallest distance in each row). Resorting into two clusters,

$$\left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\} \quad \left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

The cluster centers are re-set as the centroids:

$$\begin{bmatrix} -2/3 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 3/2 \end{bmatrix}$$

After one more sort and centroid calculation, we get:

$$\begin{bmatrix} -1 \\ -1/2 \end{bmatrix}, \begin{bmatrix} 2/3 \\ 4/3 \end{bmatrix}$$

which does not change afterward (so the clusters also remain the same).

The basic algorithm may have a few shortcomings. For example,

- The centers may be (or become) empty. To avoid this initially, it is customary to initialize the cluster centers using k data points randomly chosen. If a cluster becomes empty, we might delete it or simply ignore it.

- We had to pre-define the number of clusters. Is it possible to construct an algorithm that will grow or shrink the number of clusters based on some performance measure? We'll look at this question again after some programming.

Exercises

1. Fill in the missing details from the example in the text: Fill in the EDM and the vector M after the second sort, and continue the example one more iteration to show that the clusters do not change. Hint: To compute the distance matrix, use the points as they were originally ordered.

2. Given p scalars, $x_1, x_2, \ldots, x_p$, show (using calculus) that the number μ that minimizes the function:

$$E(\mu) = \sum_{k=1}^{p} (x_k - \mu)^2$$

   is the mean, $\mu = \bar{x}$.

3. Generalize the last exercise so that the scalars (and μ) are now vectors in $\mathbb{R}^n$. That is, given a fixed set of p vectors in $\mathbb{R}^n$, show that the vector μ that minimizes:

$$E(\mu) = \sum_{k=1}^{p} \|\mathbf{x}^{(i)} - \mu\|^2$$

   is the mean (in $\mathbb{R}^n$).

4. Write the following as a Matlab function. The abbreviation EDM is for Euclidean Distance Matrix. Good programming style: Include comments!

```
function z=edm(w,p)
%  A=edm(w,p)
%  Input:  w, number of points by dimension
%  Input:  p is number of points by dimension
%  Ouput:  Matrix z, number points in w by number pts in p
%          which is the distance from one point to another

[S,R] = size(w);
[Q,R2] = size(p);
p=p';
if (R ~= R2), error('Inner matrix dimensions do not match.'),end

z = zeros(S,Q);
if (Q<S)
  p = p';
  copies = zeros(1,S);
  for q=1:Q
    z(:,q) = sum((w-p(q+copies,:)).^2,2);
  end
else
  w = w';
  copies = zeros(1,Q);
```

```
    for i=1:S
      z(i,:) = sum((w(:,i+copies)-p).^2,1);
    end
  end
  z = z.^0.5;
```

5. Write a Matlab function, `lbgUpdate` that takes in a $p \times n$ data set X (p points in $\mathbb{R}^n$), a matrix of cluster centers $C^{k \times n}$, representing k centers in $\mathbb{R}^n$. Output the updated cluster centers, and a vector containing the distortion error on each cluster.

   Test your code using our toy data set. You might find it convenient to write a script file that sets up the data and calls `lbgUpdate` in a loop.

6. Will the cluster placement at the end of the algorithm be independent of where the clusters start? Answer this question using all the different possible pairs of initial points from our toy data set, and report your findings- In particular, did some give a better distortion error than others?

Growing and Shrinking the LBG

We can think about ways to prune and grow the number of clusters, rather than making it a predefined quantity. Here are some suggestions for modifications you can try:

- Split that cluster with the highest distortion measure, and continue to split clusters until the overall distortion measure is below some preset value. The two new cluster centers can be initialized a number of ways- Here is one option:
  $$\boldsymbol{c}^{(i_1,i_2)} = \boldsymbol{c}^{(i)} \pm \boldsymbol{\epsilon}$$
  However, this may again lead to empty clusters.

- We can prune away clusters that either have a small number of points, or whose distortion measure is smaller than some pre-set quantity. It is fairly easy to do this- Just delete the corresponding cluster center from the array.

9.3 K-means Clustering via SVD

Ah- The SVD. Is there nothing that it cannot do?

Relatively new to the area of clustering, a technique known as *spectral clustering* brings a new development: We can perform $k-$means clustering (a.k.a. LBG) by using the SVD. The general idea is that, given p data points in $\mathbb{R}^n$, we compute the $p \times p$ similarity matrix (we will use the Euclidean Distance Matrix).

"K-means clustering via Principle Component Analysis", by Ding, Chris and He, Xiaofeng, 2004 (Lawrence Berkeley National Lab)

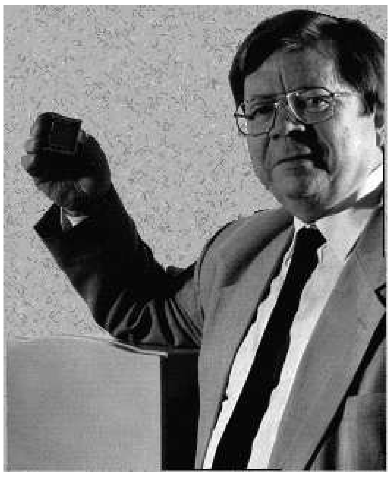

Figure 9.4: Teuvo Kohonen (from his web site) holding a well deserved drink.

9.4 Kohonen's Map

Teuvo Kohonen (See Figure 9.4) is a Neural Network researcher at the Helsinki University of Technology in Finland. He has made his code available to the public at his programs web site

`http://www.cis.hut.fi/nnrc/nnrc-programs.html`

Kohonen's Map is important in several ways. The first is that the cluster centers *self-organize* in such a way as to mimic the density of the given data set, but the representation is constrained to a preset structure. We'll see how that works later. Secondly, Kohonen is convinced that this map is a simple model on how neurons in the brain can self-organize. To read more about this, see Kohonen's book [20]. Thirdly, the general principles of using this map can be generalized to solve other problems. Again, to read more about this, consult Kohonen's book. We will focus here on the clustering aspects of Kohonen's Map (a.k.a. Self-Organizing Map, or SOM).

In biological arrays of neural cells, if one cell is excited, it will dominate the array's response to a given signal. Nearby cells may give a weak response, while cells that are far away give no response at all. One can see this "on-center, off-surround" concept in arrays of visual cells, for example. In Figure 9.5, we illustrate this concept that is also known as **competition**. Here we see a rectangular array of cells. In this case, the winning cell is at the $(3, 1)$ position. Its *receptive field* is approximately 1.5 units. Outside that field, the cells are not responding.

We are introducing a new structure- the cell structure or array - into the data. Therefore, we will define a cluster center in two ways: Each cluster center has a placement in $\mathbb{R}^n$, with the data, and a placement in the cell array. Typically, cells are arranged in a rectangular or hexagonal grid- although this is not a necessity- they may represent points on a sphere, or a torus, etc. The

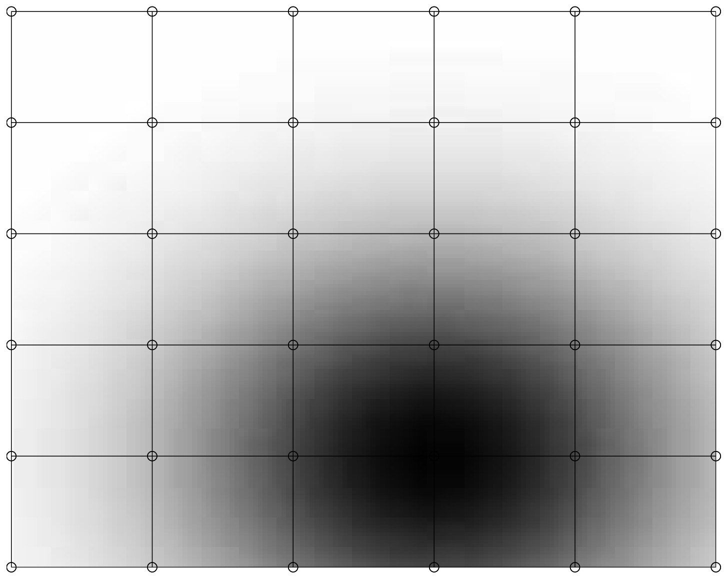

Figure 9.5: A rectangular array of cells. The grayscale value corresponds to how much the cells are responding to a given pattern. The winning cell is at position $(3, 1)$

important point here is that, whatever the structure, it is possible to obtain a metric between cluster centers. Denote the distance between centers i and j using this metric as: $d_{\mathcal{I}}(i, j)$

Throughout the training, this structure will remain fixed.

1. **Exercise:** Let cell w be fixed. How does the following quantity change for cells i, $i = 1, 2, \ldots, p$? What does the λ control?

$$\exp\left(\frac{-d_{\mathcal{I}}^2(i, w)}{\lambda^2}\right)$$

2. **Exercise:** Let $\boldsymbol{x}$ and $\boldsymbol{c}$ be vectors in $\mathbb{R}^n$. Describe what the following line of programming code does (in graphical terms):

$$\boldsymbol{c} = \boldsymbol{c} + \epsilon \cdot (\boldsymbol{x} - \boldsymbol{c})$$

3. **Remark:** The SOM, like the LBG, requires an initial placement of the cluster centers. The algorithm produces a "better" placement of the clusters. In the SOM, the training will slowly unfold the preset structure into the data space.

4. **Remark:** The SOM, unlike the LBG, does not have a known quantity that it minimizes.

5. **Update of the Training Parameters:** If α is a training parameter, we will use the following update procedure:

   - Set the initial α (α_i) and ending α, α_f. Set the maximum number of iterations, `tmax`.
   - $$\alpha(k+1) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{\frac{k}{tmax}} \tag{9.2}$$

6. **Kohonen's SOM Algorithm:**

   - Initialize centers, $\epsilon_{i,f}$, $\lambda_{i,f}$, `tmax`
   - Choose a data point $\boldsymbol{x}$ at random from X.
   - Find the closest center, $\boldsymbol{c}_w$. Call this the winning center.
   - Update all centers:

   $$\boldsymbol{c}^{(i)}(k+1) = \boldsymbol{c}^{(i)}(k) + \epsilon(k) \cdot \exp\left(\frac{-d_{\mathcal{I}}^2(i,w)}{\lambda^2(k)}\right)\left(\boldsymbol{x} - \boldsymbol{c}^{(i)}(k)\right)$$

   - Update ϵ, λ according to Equation (9.2).
   - Repeat until a stopping criterion has been attained (usually when `tmax` has been reached).

7. **Training Notes:**

   - Initially, use large values of λ (about half the diameter of the set). This strongly enforces the *ordering* of the cells.
   - Small values of λ "relaxes" the ordering, and allows the cluster centers to spread out amongst the data set.
   - Cluster centers are initialized at random.
   - Stopping criteria: There are several alternatives. One method is to stop if the centers are no longer changing significantly. Matlab will simply stop after the preset number of iterations.

8. **Matlab note:** Matlab splits the training into two distinct phases: Ordering and Training. The ordering phase allows the centers to unfold and order themselves, while the training phase puts the centers into the data set.

   Therefore, there are several parameters in Matlab's algorithm:

   - The value of λ is initially automatically set to be the maximum distance between centers, and during the ordering phase, is reduced to 1.
   - Ordering phase number of steps, q.

- Ordering phase learning rate, α. α is decreased linearly during the ordering phase, and after q steps, is equal to the training phase learning rate, below.
- Training phase neighborhood distance: This is λ in our previous example, and will remain fixed during the training phase.
- Training phase learning rate, β. During training, β decreases slowly, and the neighborhood parameter is held fixed.

9. **Kohonen's Map Example:**

   In the following example, we construct a rectangular grid of two dimensional neurons (2×3). The data set will consist of 1,000 points in $\mathbb{R}^3$, and the matrix P is 3×1000. The following commands will set up the self-organizing map and train the network. Following this set of commands, we will describe what each does.

```
1  P=rand(3,1000);
2  OLR=0.9;
3  OSTEPS=1000;
4  TLR=0.02;
5  TND=1;
6  net=newsom(minmax(P),[10,10],'gridtop','dist',
   OLR,OSTEPS,TLR,TND);
7  net.trainParam.epochs=2000;
8  net=train(net,P);
```

   If you want to use just the default parameters given above, the `newsom` command can be shortened to:

```
6  net=newsom(minmax(P),[10,10]);
```

   To plot the result (this is plotting the centers in the data space with connections, versus labeling the data in the center topology):

```
9  plotsom(net.iw{1,1},net.layers{1}.distances)
```

   To classify new data points in a data set X, we can type:

```
A=sim(net,X);
```

   The output of the network will be a sparse array type. For example, if we have a 10×10 grid array with 20 points in X, A will be a 100×20 matrix, where $A(i, j) = 1$ iff data point i is associated to cluster center j.

10. Program explanations:

- Line 1 sets up a random data set for training.
- OLR="Ordering phase learning rate"
- OSTEPS="Ordering phase number of steps"
- TLR="Training phase learning rate"
- TND="Training phase Neighborhood distance"
- The previous four items are set at Matlab's default values, which is fine for generic problems.
- Line 6 is the main command. `newsom` initializes a network structure for the SOM and inputs all of the needed training parameters. The first argument, `minmax`, returns a matrix with the minimum and maximum values of each dimension of the training set. The next argument defines the network topology (10 rows of 10 neurons each). Three numbers here would represent a three dimensional array of neurons, etc. The next argument, `gridtop`, tells the program that the array is a (two dimensional) grid. Other options are a hexagonal grid (the default- use `hextop`, or random placement, `randtop`. The next command defines the metric to use. In this case, we are using the standard Euclidean metric, but other possibilities are `boxdist` (square neighborhoods) or `linkdist` (which is the default), which uses the number of links between cells i and j. The next four options were defined previously.

11. **Kohonen's Map and Density** To further contrast Kohonen's Map with the LBG, the SOM will attempt to "mimic" the density of the data.

12. **Matlab Project: Taxonomy** In this project, we use Kohonen's Map to visualize high dimensional data in two dimensions. We will compare this to the Karhunen-Loéve projection to two dimensions.

What's this project about?

In many applications, we are given high dimensional data that we would like to visualize in two dimensions. In this project, the goal of the clustering algorithm is to see which groups of data points belong together - That is, how is the data sitting in that high dimensional space.

One application that we look at here is animal taxonomy, where we group animals together based on their physical characteristics. Another application that Kohonen and his group are working on is to classify documents on the Web according to their similarities.

Description of the data

The data in `tax001.dat` is a matrix that is 13×16 with entries either 0 or 1. Each column represents characteristics of one animal, 0 means that characteristic is not present, 1 means that the characteristic is present. The characteristics are (in order):

- Is small, medium, big (first three entries)
- Has 2 legs, 4 legs, hair, hooves, mane, feathers (next 6 entries)
- Likes to fly, run, fly, swim (next 4 entries)

The animals (in order) are: Dove, Hen, Duck, Goose, Owl, Hawk, Eagle. Fox, Dog, Wolf. Cat, Tiger, Lion. Horse, Zebra, Cow. (Grouping with periods was for clarity).

The m-file (script) `taxnames.m` will load the animal names into a cell array. For example, after running the script, you should find the variable `names`. If you type `names{1}`, Matlab will return string one, which is `Dove`. This will be handy in the plotting routines below.

NOTE: Cell arrays are handy to use if you want to store a series of vectors that are not of the same size. They can also be used to store other types of data, and by using strings, one can also use a cell array to index a family of functions (by their m-file names).

Project, Part I: For the 16 data points in $\mathbb{R}^{13}$, perform a KL projection to the best two dimensional space. On the two dimensional plot of the 16 data points, label them according to the animal name.

In Matlab, you can plot a text string at coordinates (x, y) by using the `text` command. For example:

```
x=rand(16,1);
y=rand(16,1);
text(x,y,names);
```

will plot the names of the animals (if you have already run `taxnames`).

Project, Part II: Use Matlab's SOM commands to map the 16 data points onto a 10×10 rectangular array of neurons. Plot the resulting classifications on the rectangular grid to see if you get results that are similar to Kohonen's (see attached page). I will give you a program that will perform this plotting (Matlab does not have one that is ready-made). Kohonen used a total of $2,000$ presentations.

9.5 Neural Gas

Before we begin, let us summarize the clustering procedures that we have up to this point. The LBG algorithm performs a simple clustering of the data, putting cluster centers into the data. Kohonen's map adds another element, a topological structure on the cluster centers. This is useful if we have a topology in mind (i.e., a rectangular grid for two dimensional visualization of high dimensional data). What if we don't know what topological structure to use?

This question can be re-stated as: Find a *topology preserving* clustering.

1. **Definition:** A clustering is said to be *topology preserving* if it maps neighboring cells from the topology to neighboring clusters in $\mathbb{R}^n$, and neighboring data points in $\mathbb{R}^n$ to neighboring cells in the topology.

   In Figure 9.6, we see three topologies mapped to the data, which is a uniform distribution of points in $\mathbb{R}^2$. In the first picture, the topology of the cells is a one dimensional set. In this situation, the cluster mapping is not topology preserving, because neighboring cells in the topology are not adjacent in the plane. In the second situation, we have a three-dimensional topology mapping to the plane. In this case, neighboring data points in $\mathbb{R}^2$ are mapped to non-neighboring cells in the topology. Only in the third picture do we see that both parts of the topology preserving mapping are satisfied.

2. **Remark:** The Neural Gas Algorithm [23, 25, 24] was constructed to do away with an *a priori* topological structure. It will build the topology as it clusters.

3. **Definition:** To define the topology, we need to construct a Connection Matrix, M, where

$$M_{ij} = \begin{cases} 1 & \text{If cell i connected to j} \\ 0 & \text{Otherwise} \end{cases}$$

   so that M, together with the center positions in $\mathbb{R}^n$, form the cluster topology.

4. **Remark:** Critical to the topology mapping in Kohonen's SOM was the metric: $d_{\mathcal{I}}(i, w)$. We'll need something else to take its place.

5. **Definition: The Neural Gas metric**

   As before, define w as the index of the closest center to a given data point $\boldsymbol{x}$. Sort the centers according to their distances to $c^{(w)}$, and put the ordered indices into a vector, $V = \{w, i_1, i_2, \ldots, i_{k-1}\}$. Therefore, $V(k)$ represents the index of the k^{th} closest center to $C^{(w)}$. Then:

$$d_{ng}(i, w) = k - 1$$

   where $V(k) = i$. So, $d_{ng}(i, w)$ counts how many centers are closer to w than center i is.

6. **Example:** Let $C = 0.1, 0.2, 0.4, 0.5$. If $x = 0.25$, then $w = 2$, and $V = \{2, 1, 3, 4\}$. And, $d_{ng}(1, 2) = 1$, $d_{ng}(2, 2) = 0$, $d_{ng}(3, 2) = 2$, $d_{ng}(4, 2) = 3$.

7. **Definition: Connection Update:** The Neural Gas algorithm will also be constructing the connection matrix. The main idea is the following:

   > "Let $c^{(w)}$ be the center closest to data point $\boldsymbol{x}$, and let $c^{(k)}$ be its closest center. Then set $M_{w,k} = 1$."

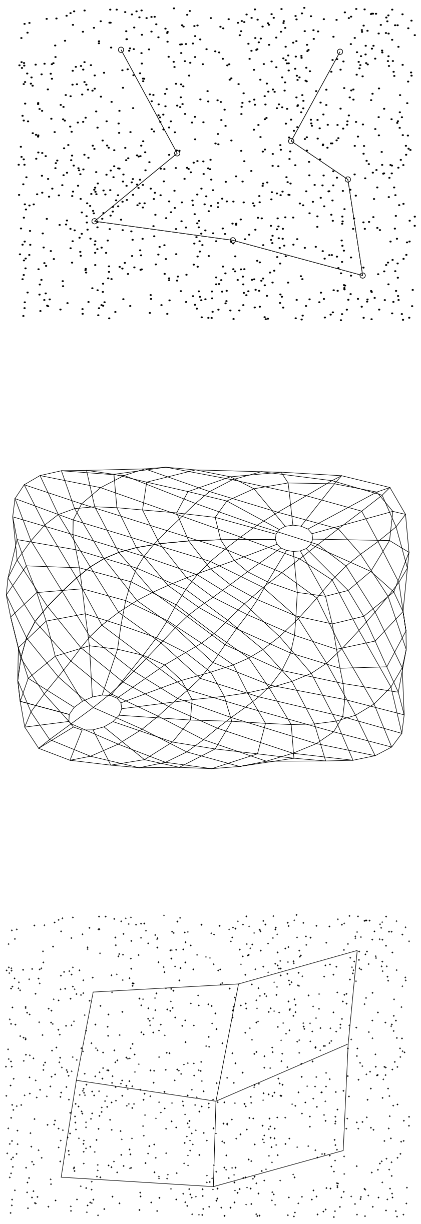

Figure 9.6: Which mapping is topology preserving? Figure 1 shows a mapping that is not topology preserving, since neighboring cells are mapped to non-neighboring points in $\mathbb{R}^2$. On the other hand, the middle clustering is not topology preserving, because neighboring points in $\mathbb{R}^2$ are mapped to non-neighboring cells in the topology. Only the third picture shows a topology preserving clustering.

Problem: As centers get updated, the closest neighbors will change.

Solution: We will keep track of the age of the connections and remove them if the have aged past a preset criteria, T^M.

That is, we will construct a time matrix T where T_{ij} =age since $M_{i,j}$ was last updated.

8. **Remark:** Notice that a connection matrix can be constructed independently of the cluster center updates.

9. **The Neural Gas Algorithm:**

   (a) Initialize the centers, M, T, $\epsilon_{i,f}$, $\lambda_{i,f}$, $T^M_{i,f}$ `tmax` (max number of iterations for NG). For example, in [23, 25, 24], they used: (N=number of data points)

   - $\epsilon_i = 0.3$, $\epsilon_f = 0.05$,
   - $\lambda_i = 0.2N$, $\lambda_f = 0.01$,
   - $T^M_i = 0.1N$, $T^M_f = 2N$
   - `tmax`$= 200N$.

   The parameters with the i, f subscript are updated using Equation (9.2) described in the last section.

   (b) Select a data point $\boldsymbol{x}$ at random, and find the winner, $c^{(w)}$.

   (c) Compute V.

   (d) Update all centers:

   $$c^{(i)} = c^{(i)} + \epsilon \exp\left(\frac{-d_{ng}(i,w)}{\lambda}\right)(\boldsymbol{x} - c^{(i)})$$

   (e) Update the Connection and Time Matrices: Let $i = V(2)$. If $M_{w,i} = 0$, set $M_{w,i} = 1$, and $T_{w,i} = 0$. If $M_{w,i} = 1$, just reset $T_{w,i} = 0$. Age all connections by 1, $T_{j,k} = Tj, k + 1$.

   (f) Remove old connections. Set $M_{w,j} = 0$ whose corresponding entries in T are larger than the current value of T^M.

   (g) Repeat.

10. **Results of the Neural Gas Algorithm** It is shown in [24] that the Neural Gas algorithm produces what is called an *induced Delaunay Triangulation.* A Delaunay Triangulation is what we produced in the introduction of this chapter when we were building a Voronoi Diagram. The induced triangulation will be a subgraph of the full triangulation. See [24] for more details on the definitions.

11. **A path preserving mapping of a manifold.** The Neural Gas algorithm has also been used to construct a discrete, path preserving representation of an environment. See the attached picture, from [24].

12. **Other applications.** Later, we will use the Neural Gas algorithm to perform time series predictions.

9.5.1 Matlab and Neural Gas

Matlab has not yet implemented a version of the Neural Gas algorithm. We will construct a suite of programs below. Although its not necessary, we will use data structures so that we're familiar with their useage when we get to the neural networks section.

The suite of programs will be: `NeuralGas`, `initng`, `paramUpdate`, and `plotng`.

1. The program `initng` is a program that will set all of the training parameters, and initialize the centers. This program should be edited at the beginning of a problem. The code is given below. Note the presence of the data structure, which allows us to wrap the parameters and the data together.

```
function C=initng(X)
[m,n]=size(X);
C.NC=500;              %Number of clusters
C.lr=[0.3 0.05];      %initial, final learning rate epsilon
C.nbr= [0.2*n 0.01]; %initial, final lambda (neighborhood size)
C.tcon=[0.1*n 2*n];  %initial, final time  (for connection matrix)
C.tmax=200*n;         %max number of iterations
C.epochs=1;           %number of epochs (each epoch runs tmax iterations,
                        %and resets the training parameters after each.)
C.tflag=1             %Training flag: 1=Use Connection, 0=Do not use
Id=randr(n);
C.cen= X(:,Id(1:C.NC)); %Initialize the centers randomly from the data
C.M=zeros(C.NC,C.NC);   %Initialize the connection matrix (if tflag=1)
```

2. The Neural Gas main algorithm is presented below. To get things moving a little faster, one can change several things. For example, if we have a large number of centers, we probably don't need to update all of them.

```
function C=NeuralGas(X,C)
%FUNCTION C=NeuralGas(X,C)
%   This is the Neural Gas clustering algorithm.  There are
%   two ways to call this program:
%     [C,M]=NeuralGas(X);
%        With only one input argument, the program will
%        read the file initng.m for the initialization
%        procedures, and train the network.  See the bottom
%        of this file for a sample initng.m file.
%     [C,M]=NeuralGas(X,C)
%        With two input parameters, we assume the centers
%        have been initialized in the structure C.  See the
%        initng file for structure specifications.

%We will use the following as the current update parameters:
%  lr = current learning rate
%  nbr= current neighborhood size (lambda)
%  t  = current time setting (for connection matrix)

%INITIALIZATION:

if nargin==1  %use the initng file
C=initng(X);
end
```

```
[m,n]=size(X)  %Dimension is m, number of points is n
[m,numcenters]=size(C.cen)

lr=C.lr(1);
nbr=C.nbr(1);
if C.tflag
   t=C.tcon(1);
   Age=zeros(numcenters,numcenters);
end

for j=1:C.epochs
    for k=1:C.tmax
if rem(k,1000)==0
  disp('iterate =');
  disp(k)
  disp('out of')
  disp(C.tmax)
end
%****************************************************
%
%   Step 1:  Choose a data point at random and compute
%              distance vector.
%
%****************************************************

curidx=ceil(n*rand);
Nx=X(:,curidx);
D=(C.cen - repmat(Nx,1,numcenters)).^2;
dd=sum(D);
[Md,w]=min(dd);

D=(C.cen - repmat(C.cen(:,w),1,numcenters)).^2;
dd=sum(D);
[Md,Id]=sort(dd);

%*******************************************************
%
%  Step 2:  Update all centers and training parameters
%
%********************************************************

for s=1:numcenters
aa=lr*exp(-(s-1)/nbr);
C.cen(:,Id(s))=C.cen(:,Id(s))+aa*(X(:,curidx)-C.cen(:,Id(s)));
end

lr=paramUpdate(C.lr,k,C.tmax);
nbr=paramUpdate(C.nbr,k,C.tmax);

%**********************************************************
%
%  Step 3:  If necessary, update Connection matrix and Time.
%
%**********************************************************
if C.tflag
   ab=Id(2);
   C.M(w,ab)=1;
   Age(w,ab)=0;
   Ix=find(C.M>0);
   Age(Ix)=Age(Ix)+1;
   Iy=find(Age>=t);
   C.M(Iy)=0;

   t=paramUpdate(C.tcon,k,C.tmax);
end  %End of time and connection update

    end %End of C.tmax (k) loop
end %End of C.epochs (j) loop
```

3. The function `paramUpdate` is simply a call to Equation (9.2).
4. The function `plotng` is an example of how we can plot the edges of the graph that is produced using the algorithm. As presented, it only plots the two dimensional graph.

```
function cidx=plotng(C)
%Plots the connections in C as line segments.
%  C is the center structure that is constructed
%    by the neural gas routine.  cflag returns 1
%    if there are no connections.

[m,n]=size(C.M);  %A square matrix
cidx=0;

for i=1:n
  for j=1:n
    if C.M(i,j)==1
      cidx=cidx+1;
      Lx(:,cidx)=C.cen(:,i);
      Ly(:,cidx)=C.cen(:,j);
    end
  end
end

if cidx==0
   disp('No connections');
end

for i=1:cidx
  line([Lx(1,i) Ly(1,i)],[Lx(2,i), Ly(2,i)]);
  if i==1
    hold on
  end
end
```

9.5.2 Project: Neural Gas

This project explores one application of triangulating a data set: Obtaining a path preserving representation of a data set.

For example, suppose we have a set of points in $\mathbb{R}^2$ that represents a room. We have a robot that can traverse the room - the problem is, there are obstacles

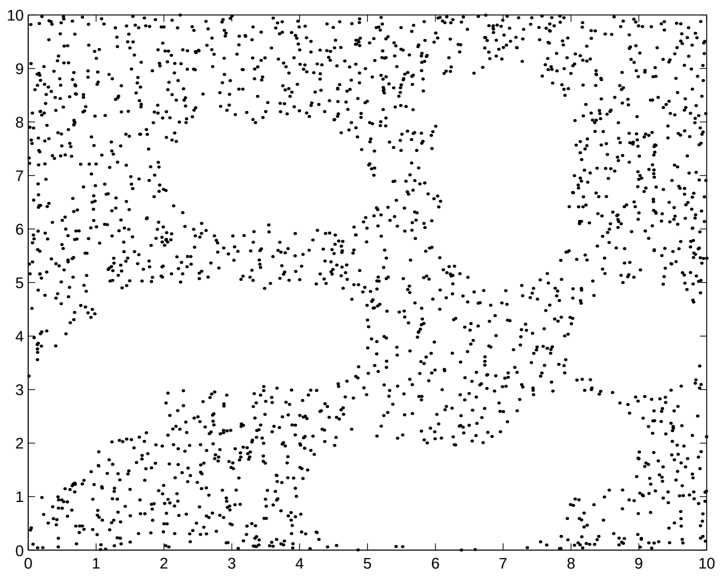

Figure 9.7: The data set that represents the "free" area. White regions indicate obstacles.

in its path. We would like a discretization of the free space so that the robot can plan the best way to move around the obstacles.

The data set is given in `obstacle1.mat`, and is plotted below. The areas where there are no points represent obstacles. **Assignment:** Use the Neural Gas clustering using $1,000$ data points and 300 cluster centers to try to represent the obstacle-free area. I would like for you to turn in:

- A printout of `initng` that you have edited for the problem.
- A printout of the results of using Matlab's profiler on the Neural Gas algorithm (you need only the report on the Neural Gas function itself, not all of its subfunctions).
- A plot (using `plotng`) of the triangulation.

9.6 Clustering and Local KL

In many applications of clustering, we look at clustering as a preprocessing of the data to perform local modeling. Before we continue, let us define some mathematical terms that will make our goals a little clearer.

1. **Definition:** A k-manifold in $\mathbb{R}^n$ is a set that is locally homeomorphic to $\mathbb{R}^k$.

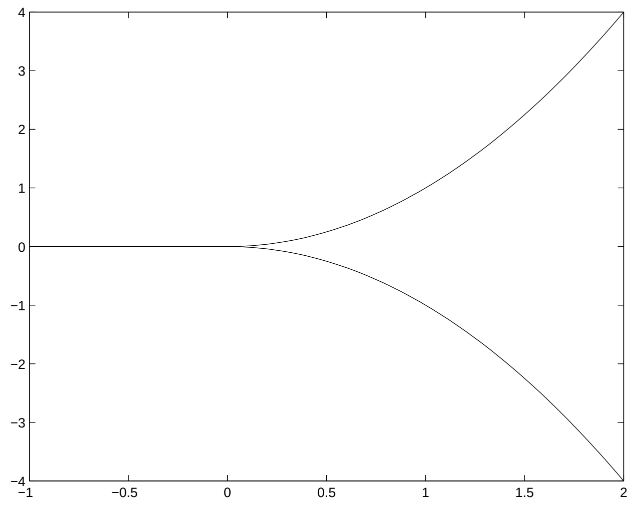

Figure 9.8: This object is not a manifold because of the branching.

2. **Examples:** The circle in the plane is an example of a 1−manifold - it is locally one dimensional. Note that it is globally two dimensional, but is the image of a one dimensional set. Similarly, the sphere in $\mathbb{R}^3$ is locally two dimensional (and is the image of a two dimensional set), but is globally three dimensional. And the graph of the function $z = f(x, y)$ is also two dimensional locally, but globally may require three dimensions in which to express it. In Figure 9.8, we see an example of something that is not a manifold. The problem is at the crotch of the branch, where a one dimensional set breaks into two branches. At this point, the set must be represented in two dimensions, while at every other point, the set is one dimensional. Note that this means that the k in "k−manifold" must remain constant throughout the set.

3. **Building the local maps.** Since a manifold is locally homeomorphic to $\mathbb{R}^k$, we should be able to build local homeomorphisms, each using k basis vectors. What are they?

   In Differential Geometry, if the manifold is locally represented as the graph of a differentiable function, we use the span of the columns of the Jacobian matrix - that is, the k basis vectors span the Tangent space.

   For us, the goal may be to construct the function so that the data represents the graph, so we have no function to differentiate. We will use the KL basis to approximate the Tangent space.

   Idea: To build the local coordinate systems, we will use clustering to place the origins, and KL to provide the local basis vectors.

4. **Remark:** Before we begin, lets review some things from Chapters 1 and 2. Suppose that $\boldsymbol{x} \in M$, where M is a $k-$manifold in $\mathbb{R}^n$. Let w be the index of the cluster center representing the origin, and let $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ be the basis vectors at $c^{(w)}$. Then the $k-$ dimensional approximation to $\boldsymbol{x}$ is:

$$x \approx c^{(w)} + \sum_{j=1}^{k} \alpha_j \boldsymbol{v}_j$$

and if $\{\boldsymbol{v}_j\}_{j=1}^k$ form an orthonormal basis, then:

$$[\boldsymbol{v}_1 \ldots \boldsymbol{v}_k]^T (\boldsymbol{x} - c^{(w)}) = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_k \end{bmatrix}$$

5. **Remark:** The error in the reconstruction of $\boldsymbol{x}$ is then:

$$\|\boldsymbol{x} - c^{(w)} - \sum_{j=1}^{k} \alpha_j \boldsymbol{v}_j\| = \| \sum_{j=k+1}^{n} \alpha_j \boldsymbol{v}_j \|$$

6. **Exercise:** Show that, if $\{\boldsymbol{v}_j\}_{j=1}^n$ forms an orthonormal set, then

$$\| \sum_{j=1}^{n} \alpha_j \boldsymbol{v}_j \| = \sum_{j=1}^{n} \alpha_j^2$$

7. **Exercise:** Using the previous exercise, and defining

$$P_w = [\boldsymbol{v}_{k+1} \ldots \boldsymbol{v}_n]^T$$

Show that the reconstruction error for the data point $\boldsymbol{x}$ is:

$$E_{x,w} = \|P_w(\boldsymbol{x} - c^{(w)})\|^2 = (\boldsymbol{x} - c^{(w)})^T P_w^T P_w (\boldsymbol{x} - c^{(w)})$$

8. **Remark:** We now proceed to incorporate this error measure into a data clustering algorithm.

Our Goal: Construct an optimal, locally linear, $k-$dimensional approximation to a given data set.

9. **VQPCA.** An algorithm has been recently suggested to perform the clustering called VQPCA (for Vector Quantization, Principle Component Analysis)[3]. This basic idea is this: Each cluster center will have a representation in $\mathbb{R}^n$ with the data, but will also keep a set of local basis vectors.

10. **The distance measure for VQPCA** Data point $\boldsymbol{x}$ will belong to cluster center w if:
$$E_{x,w} \leq E_{x,i} \quad \text{for all } i = 1 : p$$

11. **The VQPCA Algorithm**
    (a) Initialize the cluster centers and local basis vectors. The initial selection of which data point goes where is done by the standard membership function. Additionally, we must select a target dimension (we used k in the discussion above).
    (b) Sort the data in X using the measure $E_{x,i}$.
    (c) Reset the centers to be the centroids of the new sets of data.
    (d) Reset the basis vectors by using local KL.
    (e) Repeat until the maximum number of iterations is attained.

9.6.1 VQPCA in Matlab

Matlab does not have the VQPCA algorithm built in, so we'll have to write some code to do it. Again, we separate the code into two pieces as for Neural Gas. The first file is `initVQPCA`, and will perform the initialization process for us. The second file, `VQPCA` will perform the actual clustering.

(a) The program `initVQPCA`. Initially, we will need to set up the training parameters, set up the initial clusters, and perform local KL on the data in each cluster. Below, I have chosen to use a cell array to store the basis vectors for each cluster: `V.B{i}` is a matrix holding the KL eigenvectors for cluster i.

```
function V=initVQPCA(X)
%Initialize the structure for the VQPCA algorithm.

[n,m]=size(X);        %X is number of points by dimension.
V.numcenters=50;      %Number of clusters
V.tmax=20;            %max number of iterations
V.tdim=2;             %Target dimension

%V.C (number of centers by dimension) holds centers.
%V.B{i}= Basis vectors for ith center.
%Initialize centers by taking random data points.

Id=randr(n);
V.C= X(Id(1:V.numcenters),:);
```

[3]Vector Quantization is another name for Data Clustering, and Principle Component Analysis is another name for KL - We might use the acronym DCKL, instead!

```
clear Id;

%Initially, we sort using standard metric.
D=dist(V.C,X');
[Md,Id]=min(D);

for i=1:V.numcenters

  Id1=find(Id==i);
  lenId=length(Id1);
  if lenId>0
        Y=X(Id1,:);
    c=mean(Y);
    V.C(i,:)=c;
        H=Y-repmat(c,lenId,1);
        [V.B{i},s,u]=svd((1/lenId)*H'*H);
  else
     error('Empty cluster in initialization');
  end

end
```

(b) The next program is the main program. Note that we can call `VQPCA` to do the initialization first by using only one input argument. To track the error, `V.E` is a vector holding the distortion error for each training iteration.

```
function V=VQPCA(X,V)
%Main program for the VQPCA Algorithm.  There are two ways
%to call this program.  If the data set $X$ is the only
%input argument, then VQPCA first initializes the data
%structure V.  Otherwise, the algorithm proceeds as if
%V is the initial state.

if nargin==1
  V=initVQPCA(X);
end


for t=1:V.tmax
t                  %Just to keep an eye out for where we are
D=distVQPCA(V,X);
[Md,Id]=min(D');   %Md is an m vector containing min distances
                   %Id contains the indices of the mins.
V.E(t)=mean(Md);
for i=1:V.numcenters

  Id1=find(Id==i);
  lenId=length(Id1);
  if lenId>0
        Y=X(Id1,:);
    c=mean(Y);
    V.C(i,:)=c;
        H=Y-repmat(c,lenId,1);
        [V.B{i},s,u]=svd((1/lenId)*H'*H);
  else
     error('Empty cluster in VQPCA');
  end
```

```
end

end  %End of main loop
```

(c) Notice that we had to change the metric to use this program to `distVQPCA`. This program is included below. We had said previously that the metric we would use to measure distances for cluster w is:

$$(x - c^{(w)})^T P_w^T P_w (x - c^{(w)})$$

so that the products look like:

`(1 x n)(n x n)(n x 1)`

However, if we substitute the data point x with the mean subtracted matrix X, our product would produce a $p \times p$ matrix, Z, where:

$$Z(i,j) = (x^{(i)} - c^{(w)})^T P_w^T P_w (x^{(j)} - c^{(w)})$$

all we want from Z is its diagonal. Therefore, we want to compute the diagonal of the matrix product (assume X is center-subtracted, and has size $n \times p$):

$$X^T P_w^T P_w X$$

We call the non-standard function `diagProd` to perform this routine.

```
function D = distVQPCA(V,X)
%D is the distance matrix that is number of points
%  by number of centers.  D(i,j)=distance (using VQPCA)
%  between the ith data point and jth center.

[m,n]=size(X);  %X should be number of points by dimension
D=zeros(m,V.numcenters);
%Check target dimension:
if V.tdim==n
  return;
end

for i=1:V.numcenters
    A=V.B{i}(:,V.tdim+1:n);
        Y=X-repmat(V.C(i,:),m,1);
        D(:,i)=diagProd(Y*A,A'*Y')';
end  %End of main i loop
```

(d) In Figure 9.9, we see the result of running the VQPCA algorithm on data representing the half sphere. The target dimension in this case was 2, we used 1,000 data points (only 500 are shown), and 50 centers (4 are shown). Furthermore, the picture shows the vectors that were produced by the KL algorithm, and shows that they approximate tangent planes on the half sphere.

(e) **Exercise:** Reproduce the Figure by loading the data in `halfsphere`, copy `initVQPCA` into your own directory (so you can edit it), then run the VQPCA program. Call `plotVQPCA(V,X)` to plot the result.

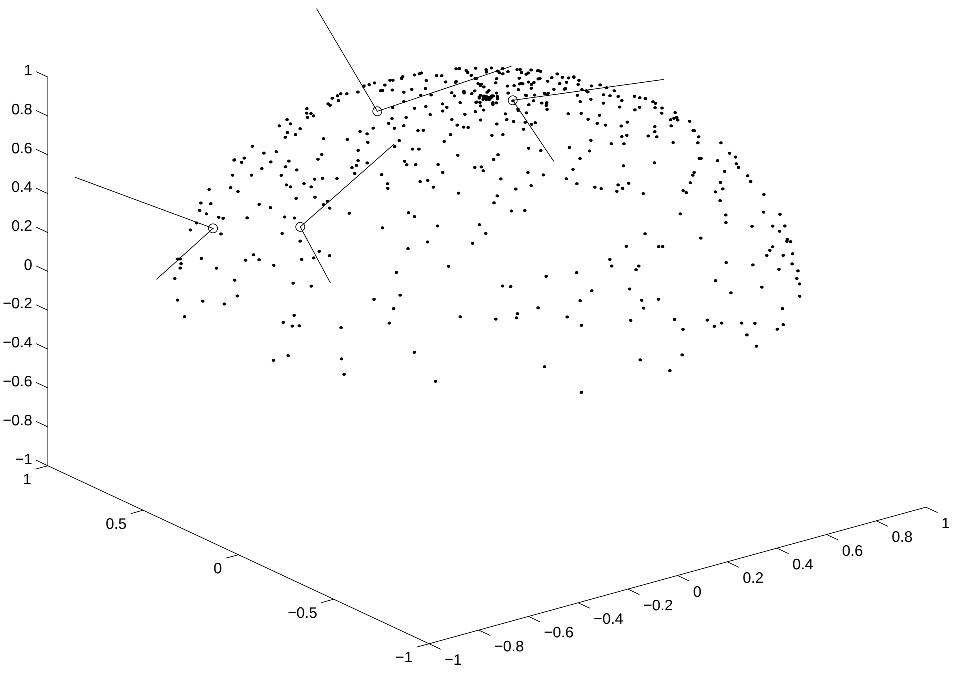

Figure 9.9: The VQPCA Algorithm applied to data representing a half-sphere. The target dimension was 2, and shown are 4 representative clusters together with their KL basis vectors (that approximate local tangent planes).

9.7 A Comparison of the Techniques

Thus far, we have looked at the LBG algorithm, Kohonen's SOM, the Neural Gas algorithm, and VQPCA. We should choose our method according to what we require for a given problem at hand. While the Neural Gas algorithm arguable gives the most information, it comes at the greatest cost. The table below summarizes the algorithms that we have considered.

Part III

Functional Representations