Data Mining

Originally, data mining was a statistician's term for overusing data to draw invalid inferences.

A famous example - David Rhine, a parapsychologist at Duke in the 1950's tested students for extrasensory perception by asking them to guess 10 cards as either red or black. He found that about 1/1000 of them guessed all 10, and instead of realizing that that is what you'd expect from random guessing, declared them to have ESP. When he retested them, he found they did no better than average. His conclusion: telling people they have ESP causes them to lose it!

Our definition will be the "The extraction of implicit, previously unknown, and potentially useful information from data"

Some famous quotes about data mining

"Drowning in Data yet Starving for Knowledge" - anonymous

"Computers have promised us a fountain of wisdom but delivered a flood of data"

William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus

"Where is the wisdom we have lost in knowledge? Where is the knowledge we have lost in information?"

T. S. Eliot

What is NOT data mining?

Data Mining, noun: "Torturing data until it confesses ... and if you torture it enough, it will confess to anything"

Jeff Jonas, IBM

"An Unethical Econometric practice of massaging and manipulating the data to obtain the desired results"

W.S. Brown "Introducing Econometrics"

Some examples of data mining

- Patterns of traveler behavior mined to manage the sale of discounted seats on planes, rooms in hotels, etc.
- The connection between diapers and beer. From the use of data mining it was observed that customers who buy diapers are more likely to by beer than average. Supermarkets then placed beer and diapers nearby, knowing many customers would walk between them. Placing potato chips between diapers and beer increased sales of all three items.
- Skycat and Sloan Digital Sky Survey - clustering sky objects by their radiation levels in different bands allowed astromomers to distinguish between galaxies, nearby stars,and many other kinds of celestial objects.
- Comparison of the genotype of people with/without a condition allowed the discovery of a set of genes that together account for many cases of diabetes. This sort of mining will become much more important as the human genome is constructed.

Data mining is an interdisciplinary field and researchers in many different areas use data mining techniques.

- Statistics
- Mathematics
- Artificial Intelligence where it is called machine learning.
- Researchers in clustering algorithms
- Visualization researchers
- Databases

Stages of Data Mining

1. Data gathering, e.g., data warehousing, web crawling
2. Data cleansing - eliminate errors and/or bogus data, e.g., patient fever = 125
3. Feature extraction - obtaining only the interesting attributes of the data, e.g., date acquired is probably not useful for clustering celestial objects
4. Pattern extraction and discovery - this is the stage that is often thought of as data mining
5. Visualization of the data
6. Evaluation of results; not every discovered fact is useful, or even true! Judgement is necessary before following your software's conclusions.

We will begin by looking at clustering to detect features of data.

Clustering

What do we mean by clustering?

- Clustering strives to identify groups/clusters of objects that behave similarly or show similar characteristics. In common terms it is also called "look-a-like groups".
- Similarity is often quantified through the use of a distance function.
- In particular application areas cluster analysis is referred to by other names; in market studies it is known as a segmentation method and in neural network concepts, it is called unsupervised learning.
- Thus clustering can be viewed as a technique which attempts to find order in a set of data. The data may be discrete or continuous.
- The data may be numerical or characters/symbols (such as genetic data).
- We will look at three types of clustering.

  - Hierarchical clustering
  - K-Means clustering
  - Geometric clustering

- We will first look at hierarchical clustering where we construct a hierarchy or tree-like structure to see the relationship among objects.
- In agglomerative hierarchical clustering the data objects are grouped in a bottom-up fashion. So we begin by having each object be its own cluster and then start grouping similar objects.
- Another approach would be divisive hierarchical clustering where the data objects are grouped in a top down method. Here we group all the objects together into a single cluster to start and then begin to separate off clusters.
- For each application, one chooses a way to determine the closeness (similarity) of clusters; often we rely on a distance function to determine how far apart groups are.
- Different hierarchical clustering methods using a different criteria for defining "closeness" between clusters.

- So the first thing we have to do is decide what we mean by distance between clusters so we can talk about closeness or similarity.
- We know how to calculate the Euclidean distance between two points in $\mathbb{R}^n$ but there are other ways to measure distance between vectors such as the infinity or max norm. However, if we want to know the distance between two clusters of points (or other objects) we need to specify what we mean.
- Often our clusters don't consist of points in $\mathbb{R}^n$; they could be colors in an image, all American made SUVs, amino acid codons (like ATG), etc.
- So we want to quantify what properties a distance function should have so that we can quantify "closeness" of clusters.

Let A, B and C be clusters (which may consist of a single object or many objects). We denote the distance between A and B as $d(A, B)$ where $d(\cdot, \cdot)$ satisfies the general properties:

1. $d(A, A) = 0$
2. $d(A, B) = d(B, A)$
3. $d(A, B) \geq 0$ (but if $d(A, B) = 0$ this does not necessary mean that $A = B$)
4. $d(A, B) + d(B, C) \geq d(A, C)$ (triangle inequality)

For example, we could use the standard Euclidean length (or any vector norm) to measure the distance between single objects but we would have to decide how to measure the closeness of clusters of points. For example, we could use nearest or farthest neighbor, average, etc. (More on this later.)

However, there are many examples of distance functions which are quite different

from our usual measures and are useful in specific applications.

Here is an example of a distance function from a word game. Suppose you have two words of length n and we want to define a path from the first word to the second word by changing only one letter at a time to make another word; the distance from one word to another is the length of the shortest path between them.

For example, let's start at "GOLF" and go to "WORD". We have

$$\text{GOLF} \implies \text{GOLD} \implies \text{COLD} \implies \text{CORD} \implies \text{WORD}$$

so if this is the shortest path then the distance between GOLF and WORD is 4. The distance between GOLF and WOLF is 1.

Note that this distance function satisfies all the properties of our metric.

Sometimes we don't have a distance function available but rather a table which measures the pairwise similarity of the objects.

Agglomerative Hierarchical Clustering

- Objects are grouped in a bottom-up fashion.
- Suppose we have N objects to cluster.
- Initially each item is its own cluster so we have N clusters.
- Our first step is to pick the two closest (in terms of how we define closeness) clusters (i.e., individual objects at this first step) using our distance function and form a cluster; we now have $N-1$ clusters and we record the information that at the first step clusters i and j were merged.
- Second we determine the closeness between the new cluster containing 2 objects and the other $(N-2)$ (which consist of single points) and merge the closest and record the fact that at the second step these two clusters were merged.
- We continue merging clusters and recording which clusters were merged.
- Termination occurs either when we have a single cluster or by a condition

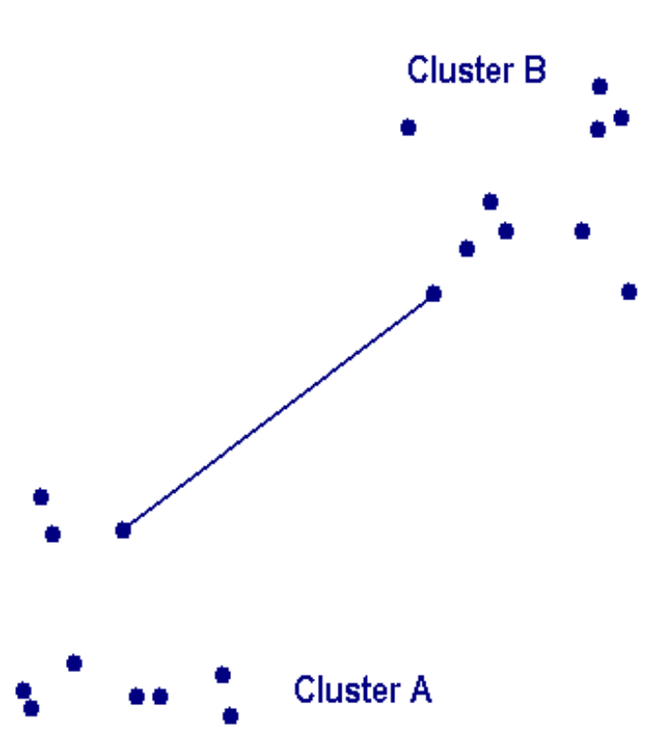
specified by the user such as a desired number of clusters or a measure of their separation.

- Different algorithms arise by using a different definition for "closeness between clusters."

Single Linkage Hierarchical Clustering

Single linkage hierarchical clustering makes a specific choice for determining the closeness between clusters.

In single linkage clustering the distance between two clusters is computed as the distance between the two closest elements in the two clusters.



Thus if cluster A consists of objects α_i, $i = 1, n$ and cluster B consists of objects β_i, $i = 1, m$ then we define the distance between A and B as

$$d(A, B) = \min d(\alpha_i, \beta_j) \quad \text{for all } i = 1, n \text{ and } j = 1, m$$

The distance between every possible object pair (α_i, β_j) is computed and the minimum value of these distances is said to be the distance between clusters A and B. In other words, the distance between two clusters is given by the value of the shortest link between the clusters.

Matlab has some built-in functions which are useful is helping us to understand hierarchical clustering; here are the commands when we want to use single linkage clustering.

- `sv = pdist ( xy )` computes the distance vector of a set of points;
- `sl = linkage ( sv, 'single' )` returns the single linkage information;
- `dendrogram ( sl )` plots the single linkage information as a tree.

The command `pdist` uses the Euclidean distance as a default but there are several other distance functions which you can specify to use.

We will look at a set of points in $\mathbb{R}^2$ and apply single linkage clustering with Matlab's help and try to interpret the results. We will just use the standard Euclidean distance to measure the distance between two points and to determine the distance between two clusters we will use the shortest link between the clusters based on the Euclidean length.

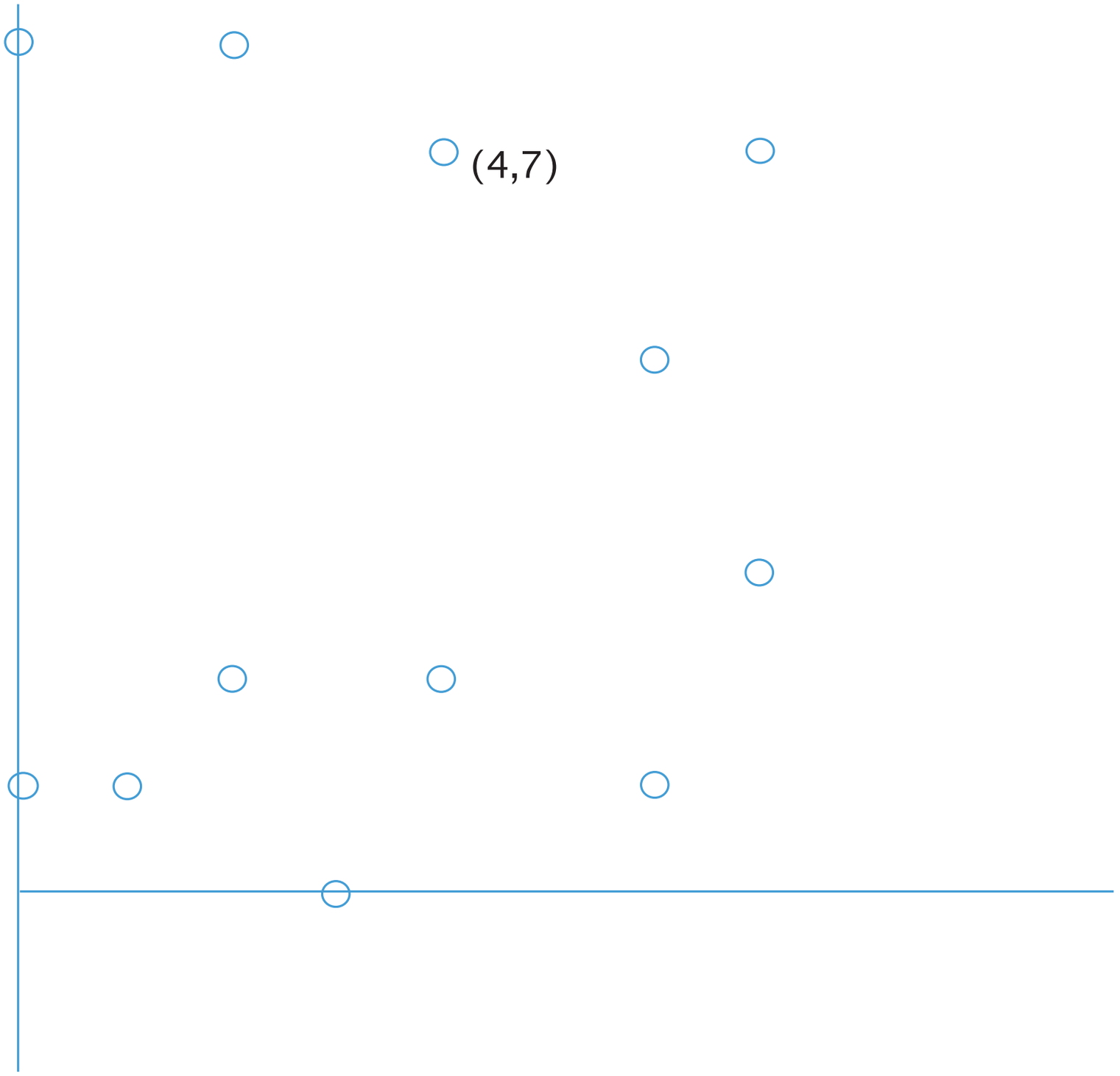
The 12 ordered points to cluster are:

$$(3,0),(0,1),(1,1),(6,1),(2,2),(4,2),(7,3),(6,5),(4,7),(7,7),(0,8),(2,8)$$

and are input into a 12 x 2 array, `XY`.

The result of the command `pdist (XY)` is a vector which gives the Euclidean distance between each pair. For example, the entries of the vector are:

```
3.1623    2.2361    3.1623    2.2361    2.2361    5.0000
5.8310    7.0711    8.0623    8.5440    8.0623    1.0000
6.0000    2.2361    4.1231    7.2801     7.2111    7.2111
9.2195    7.0000    7.2801    5.0000    1.4142    3.1623
6.3246    6.4031    6.7082    8.4853    7.0711    7.0711
4.1231    2.2361     2.2361    4.0000    6.3246    6.0828
```


(4,7)

```
 9.2195    8.0623    2.0000    5.0990     5.0000    5.3852
 7.0711    6.3246    6.0000    3.1623     3.6056    5.0000
 5.8310    7.2111    6.3246    2.2361     5.0000    4.0000
 8.6023    7.0711    2.8284    2.2361     6.7082    5.0000
  3.0000    4.1231    2.2361    7.0711     5.0990    2.0000
```

The first entry is the distance between points 1 and 2 (i.e., $\sqrt{10}$), the second entry is the distance between points 1 and 3 (i.e., $\sqrt{5}$), the eleventh entry is the distance between points 1 and 12 (i.e., $\sqrt{65} = 8.0603$) and the twelfth is the distance between points 2 and 3 (i.e.,1), etc.

The smallest distance is 1 which is the distance between points 2 and 3.

Now we form the single linkage clustering using the command `linkage` with the output from `pdist` as our input. What does this tell us? Because we have `'single'` as an argument we are asking for the single linkage clustering which

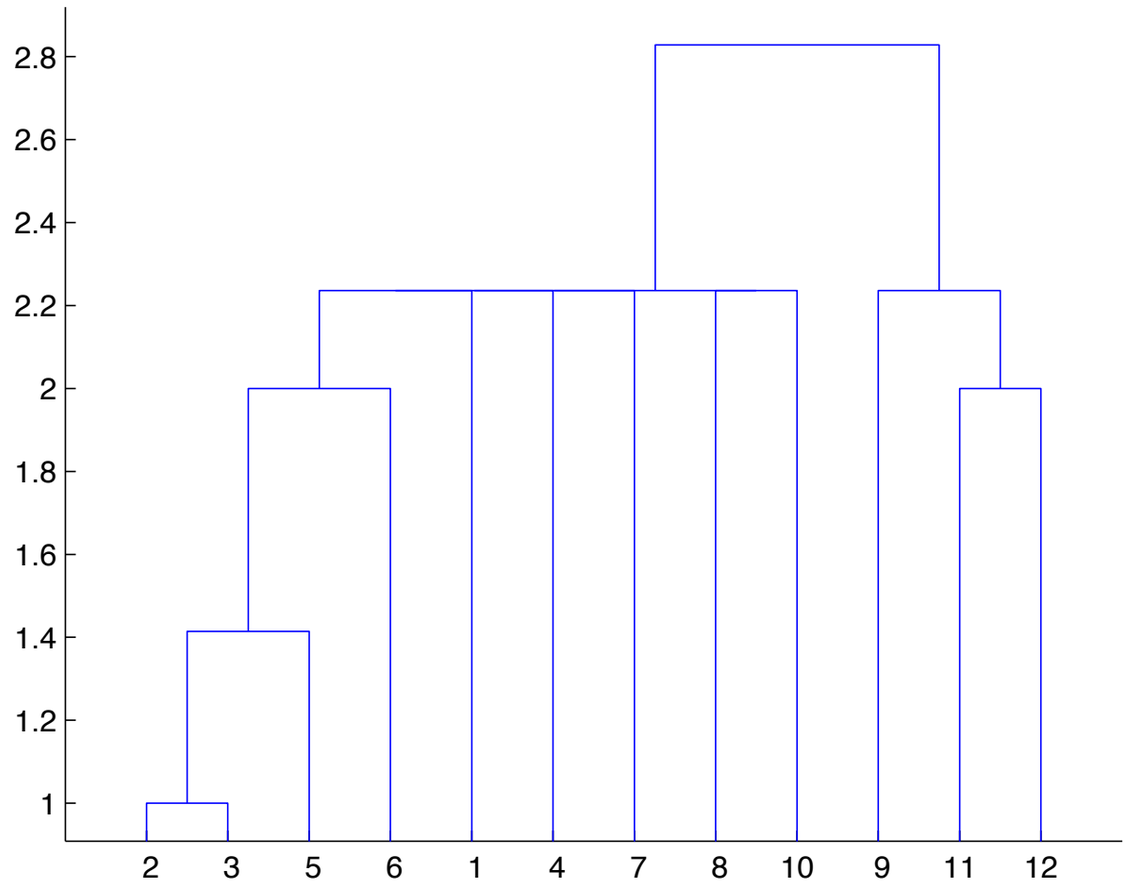
uses the nearest neighbor to link clusters; one can also use other linkage criteria such as "furthest" or "average".

For our problem, we get

```
 2.0000    3.0000    1.0000
 5.0000   13.0000    1.4142
11.0000   12.0000    2.0000
 6.0000   14.0000    2.0000
 9.0000   15.0000    2.2361
 8.0000   10.0000    2.2361
 1.0000   16.0000    2.2361
 4.0000   19.0000    2.2361
 7.0000   20.0000    2.2361
18.0000   21.0000    2.2361
17.0000   22.0000    2.8284
```

which is not too useful even in this case where we have a small amount of data. However, if we use the command to make a tree diagram the results become

clearer.



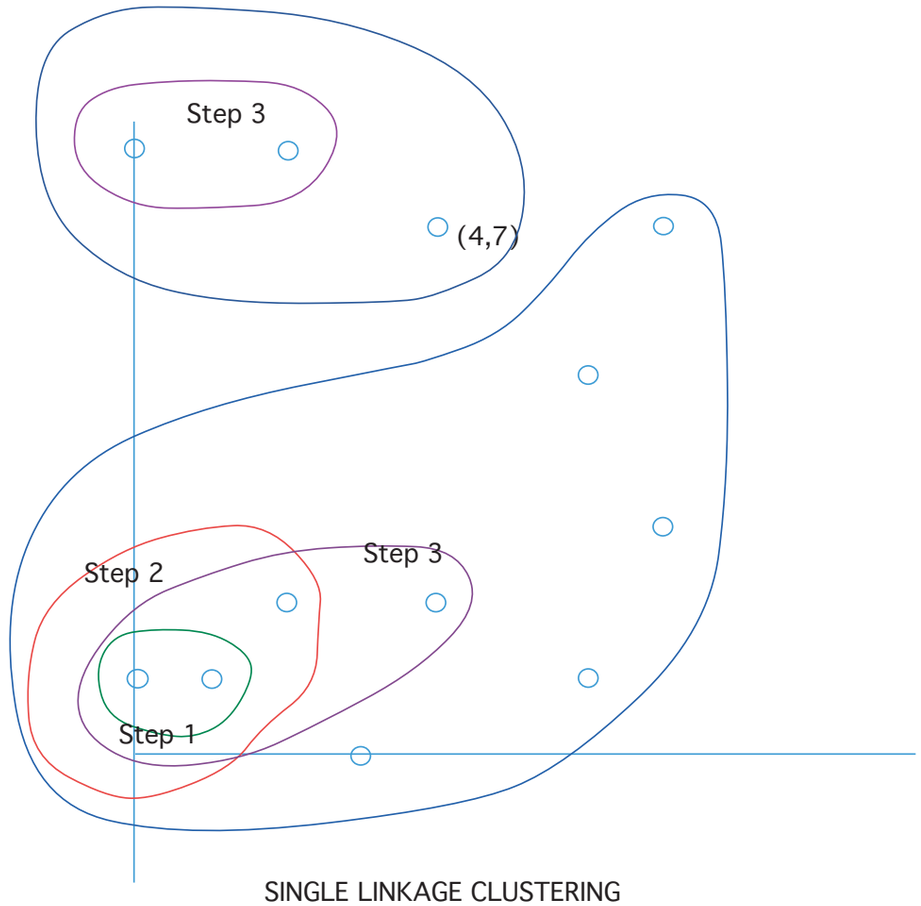
First note that Matlab has reordered the 12 points on the x-axis for readability.

For the first clustering, the closest points were # 2 - (0,1) and # 3 - (1,1) which

is illustrated by the connection on the left. Then point # 5-(2,2) was added to the cluster. The next linkage was point # 11 and # 12; then # 6 was added to the cluster containing point # 2,3,and 5; etc.

What does the y axis tell us here? If we want to group the points in clusters so that the nearest neighbor is no more than 1 unit apart, then the only candidates are points 2 and 3. If we want group the points in clusters so that the nearest neighbors are no more that $\sqrt{5} = 2.236$ then we can have two clusters where points 2,3,4,5,6,1, 7,8,and 10 make up one cluster and the points 9, 11 and 12 make up the other cluster.

For example, in the following picture we have illustrated the steps.

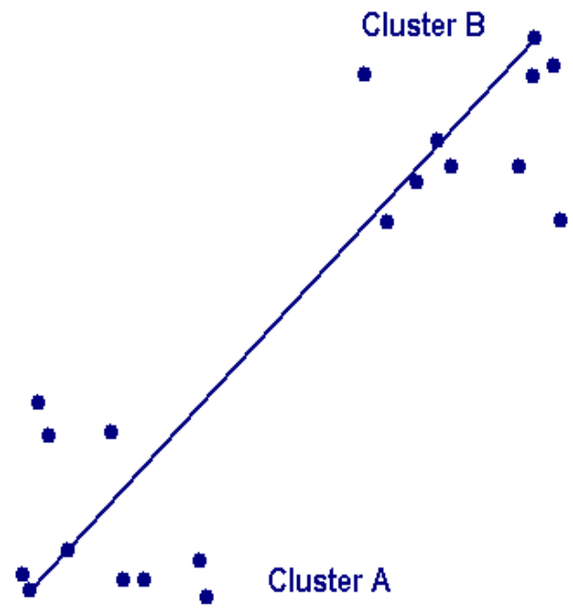


SINGLE LINKAGE CLUSTERING

A potential problem with single linkage clustering is that clusters may be forced together due to a single object. The following method eliminates this problem.

Complete Linkage Clustering or Farthest Neighbor

Complete linkage clustering, also known as farthest neighbor, is, in some sense, the opposite of single linkage. The distance between clusters is now defined as the distance between the most distant pair of objects.



Thus if cluster A consists of objects α_i, $i = 1, n$ and cluster B consists of objects

β_i, $i = 1, m$ then we define the distance between A and B as

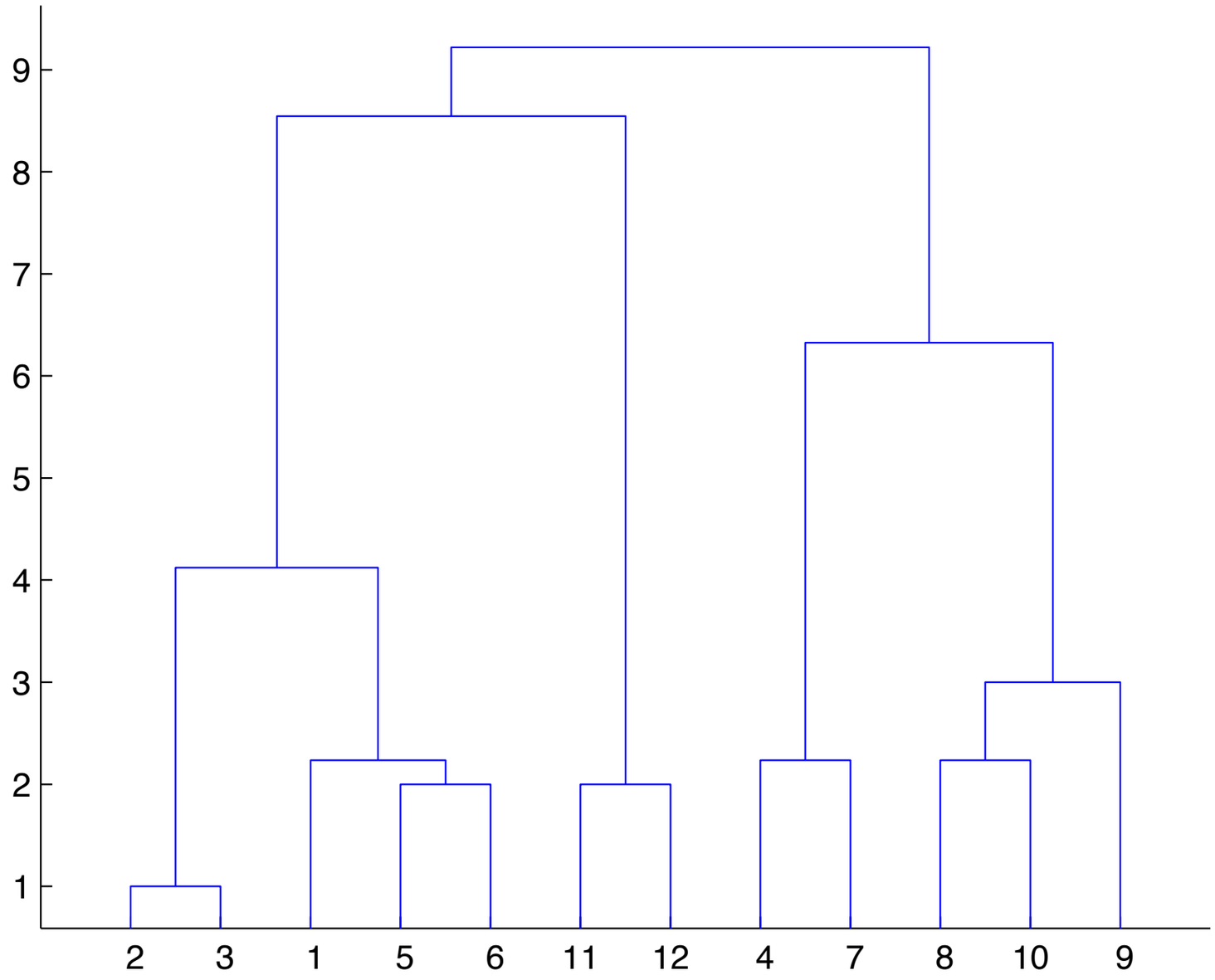
$$d(A, B) = \max d(\alpha_i, \beta_j) \quad \text{for all } i = 1, n \text{ and } j = 1, m$$

Once again the distance between every possible object pair (α_i, β_j) is computed and the maximum value of these distances is said to be the distance between clusters A and B. In other words, the distance between two clusters is given by the value of the longest link between the clusters.

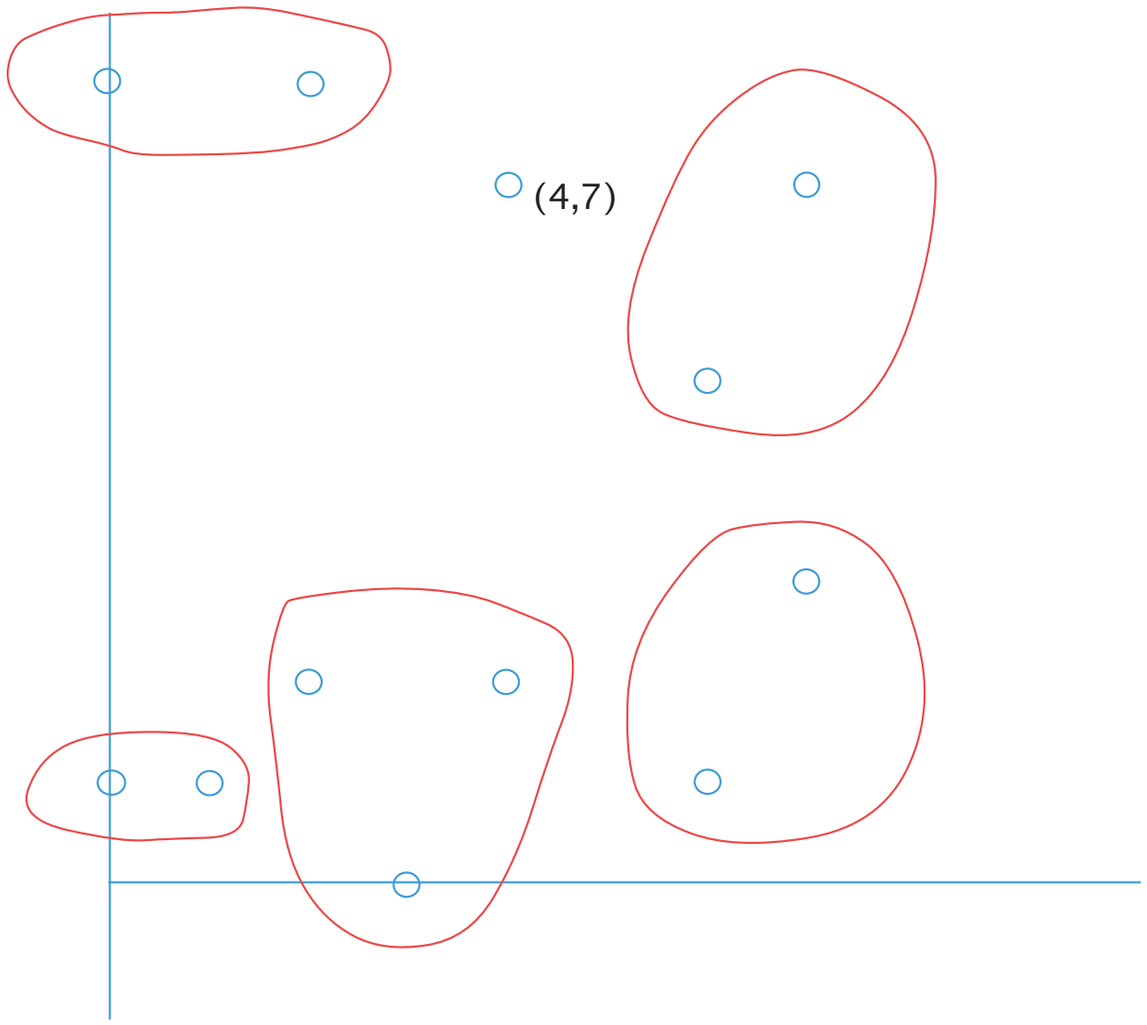
At each stage of hierarchical clustering, the clusters A and B for which $d(A, B)$ is minimum, are merged.

This means that the similarity of two clusters is the similarity of their most dissimilar members.

For our example we get the following tree diagram when we use the command `linkage(sv,'complete')` for our set of 12 points.


9
8
7
6
5
4
3
2
1
2
3
1
5
6
11
12
4
7
8
10
9

We have illustrated the clusters that are within $\sqrt{5}$.



COMPLETE LINKAGE CLUSTERING

There are other types of hierarchical clustering where we define the closeness of clusters in different ways. We summarize some of these below. Assume that cluster A consists of objects α_i, $i = 1, n$ and cluster B consists of objects β_i, $i = 1, m$. All use the criteria of merging clusters which have the minimum distance between clusters.

1. Single linkage clustering

$$d(A, B) = \min d(\alpha_i, \beta_j) \quad \text{for all } i = 1, n \text{ and } j = 1, m$$

2. Complete linkage clustering

$$d(A, B) = \max d(\alpha_i, \beta_j) \quad \text{for all } i = 1, n \text{ and } j = 1, m$$

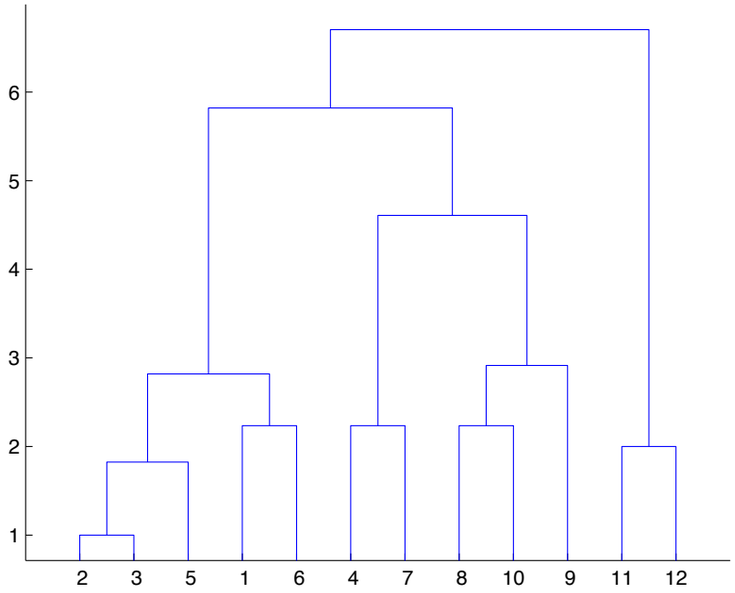
3. Average linkage clustering

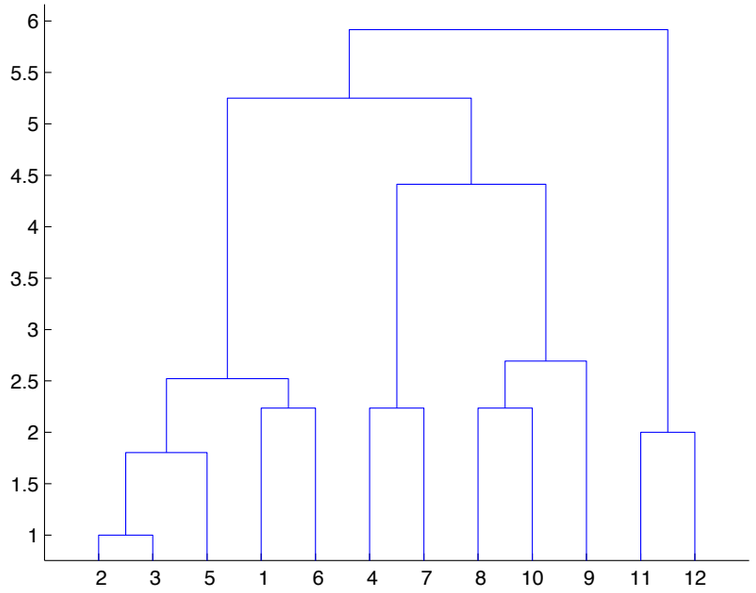
$$d(A, B) = \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} d(\alpha_i, \beta_j)$$

4. Centroid linkage clustering

$$d(A, B) = \|\bar{\alpha} - \bar{\beta}\|_2 \quad \text{where } \bar{\alpha} = \frac{1}{n} \sum_{i=1}^{n} \alpha_i \text{ and } \bar{\beta} = \frac{1}{m} \sum_{i=1}^{m} \beta_i$$

For our example, here are the tree diagrams using Average linkage clustering (left) and Centroid linkage clustering (right).





Exercise Consider the following set of points in $\mathbb{R}^2$

$$\{(1,2),(-4,-2),(3,3),(5,6),(-3,-1),(-5,-3),(5,4),(6,8),(-5,-2),(-1,-2)\}$$

a. Cluster these points using single linkage (nearest neighbor) clustering using the ℓ_2 norm. Draw a dendrogram tree and interpret your results.

b. Repeat (a) using the '`cityblock`' option for distance in the `pdist` command. What norm is this? Why do you think it is called this?

c. Repeat (a) using complete linkage (farthest neighbor) clustering.

- Hierachical clustering is useful when we think our data has a family tree relationship.
- K-Means is a different type of clustering tool which we will look at now.
- Basically we are going to look for K average or mean values about which the data can be clustered.
- We are not as interested in finding a family history but rather breaking our data into K groups.
- For example, new voting districts will be set using the 2010 census so we might want to break neighborhoods into a fixed number of groups.
- We first quantify what we mean by averages and associated energies.

Averages: Minimizing Energies

We often take the average as a representation of a set of numerical data. We have a sense of what it means but let's quantify it mathematically. We will view the average as minimizing an energy:

Lemma. Let $\bar{x}$ be the average of a set $X = \{x_1, x_2, \ldots, x_n\}$. Then $\bar{x}$ is the unique number which minimizes the energy

$$\mathcal{E}(c, X) = \frac{1}{2}\sum_{i=1}^{n}(c - x_i)^2 \text{ that is, } \bar{x} = \min_{c \in \mathbb{R}^1} \frac{1}{2}\sum_{i=1}^{n}(c - x_i)^2$$

Proof. We take the first derivative of $\mathcal{E}$ and set it to zero to get

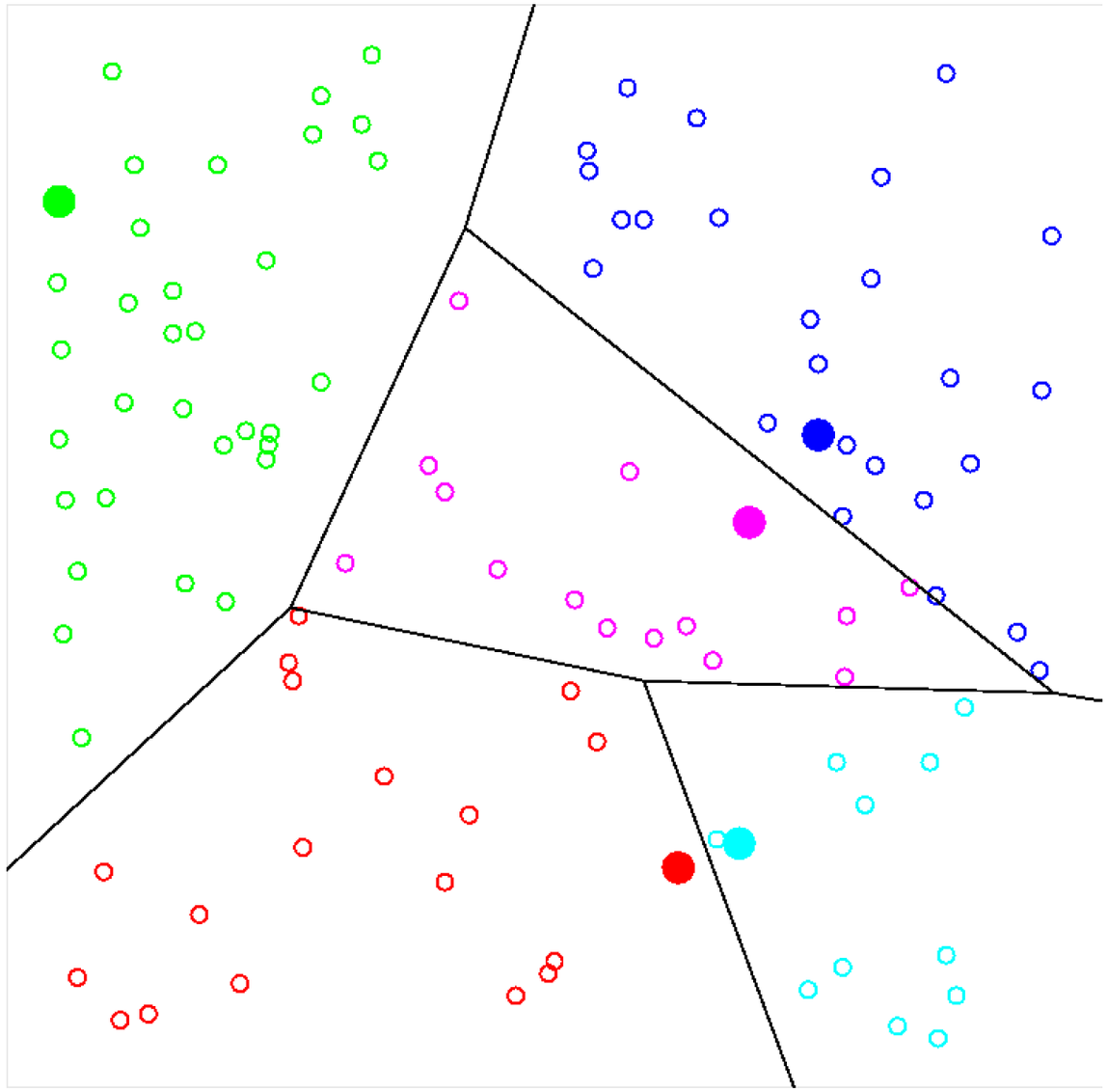
$$\frac{\partial \mathcal{E}(c, X)}{\partial c} = \frac{1}{2}\sum_{i=1}^{n} 2(c - x_i) = 0 \implies \sum_{i=1}^{n} c = \sum_{i=1}^{n} x_i \implies c = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Now that we recognize that the average minimizes an energy function, we can generalize this concept.

Assume that we have a set of K centers c_j (instead of just c) and we have more than one "average". We want to divide the data into K clusters C_j and find the points c_j such that the energy

$$\mathcal{E} = \sum_{j=1}^{K} \left[\sum_{x_i \in C_j} \|c_j - x_i\|_2^2 \right]$$

is minimized. This is the basis of K-Means clustering.



Here we have 100 points and 5 random "centers" c_j. We cluster the points by determining which of the 5 "centers" each point is closest to. The problem is, these points are not centers of the clusters and don't minimize our energy.

K-Means or Lloyd's Method

To understand K-Means we start with a set of N points in $\mathbb{R}^d$, $X = \{\vec{x}_i\}_{i=1}^N$, which we want to cluster into K groups.

Goal: Find K points c_j such that the energy

$$\mathcal{E} = \sum_{j=1}^{K} \left[\sum_{x_i \in C_j} \|c_j - x_i\|^2 \right]$$

is minimized. Here C_j consists of all points which are closer to c_j than any other c_i, $i \neq j$.

For the initialization step we choose K points called centers or generators and denote them by $\vec{c}_i$, $i = 1, \ldots, K$. These can be random but we will see that this is not always the best approach.

Step 1 For each point $\vec{x}_i$, determine $\vec{c}_j$ such that

$$\|\vec{x}_i - \vec{c}_j\| \leq \|\vec{x}_i - \vec{c}_n\| \quad \text{for all } n = 1, \ldots, N$$

Then the first cluster consists of all points which are closer to $\vec{c}_1$ then any other generator; in general cluster k consists of all points closer to $\vec{c}_k$ then any other generator.

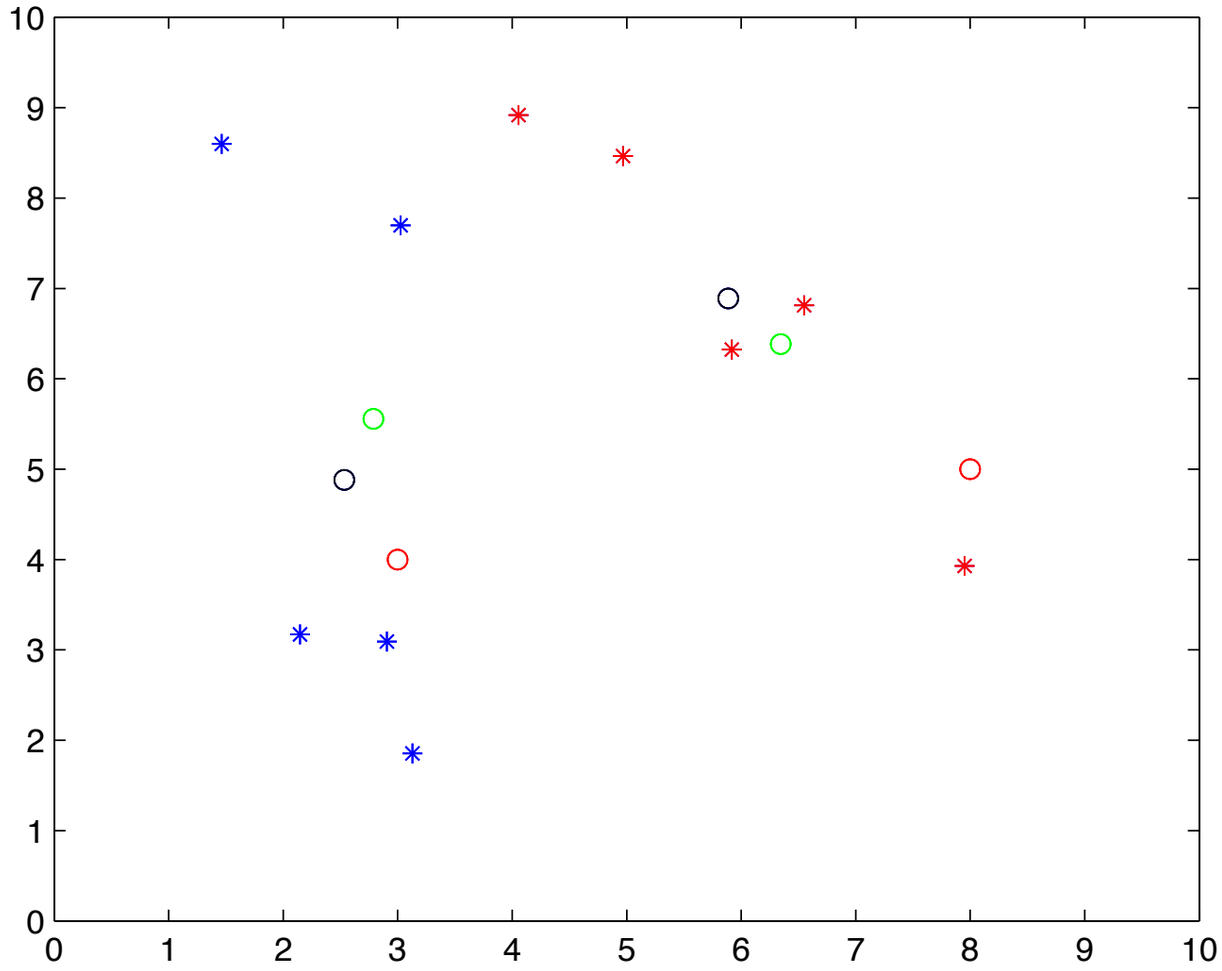
Of course, we have no expectation that these c_k, $k = 1, \ldots, K$ minimize our energy. In fact, when we plot these we see that the $\vec{c}_k$ don't even look like the centers of the clusters.

So we construct an iterative process (Lloyd's algorithm) where we move the $\vec{c}_i$ to the center of each cluster and start again. We hope that the method converges to a set of $\vec{c}_i$ which minimizes our energy but we need to see if it always does.

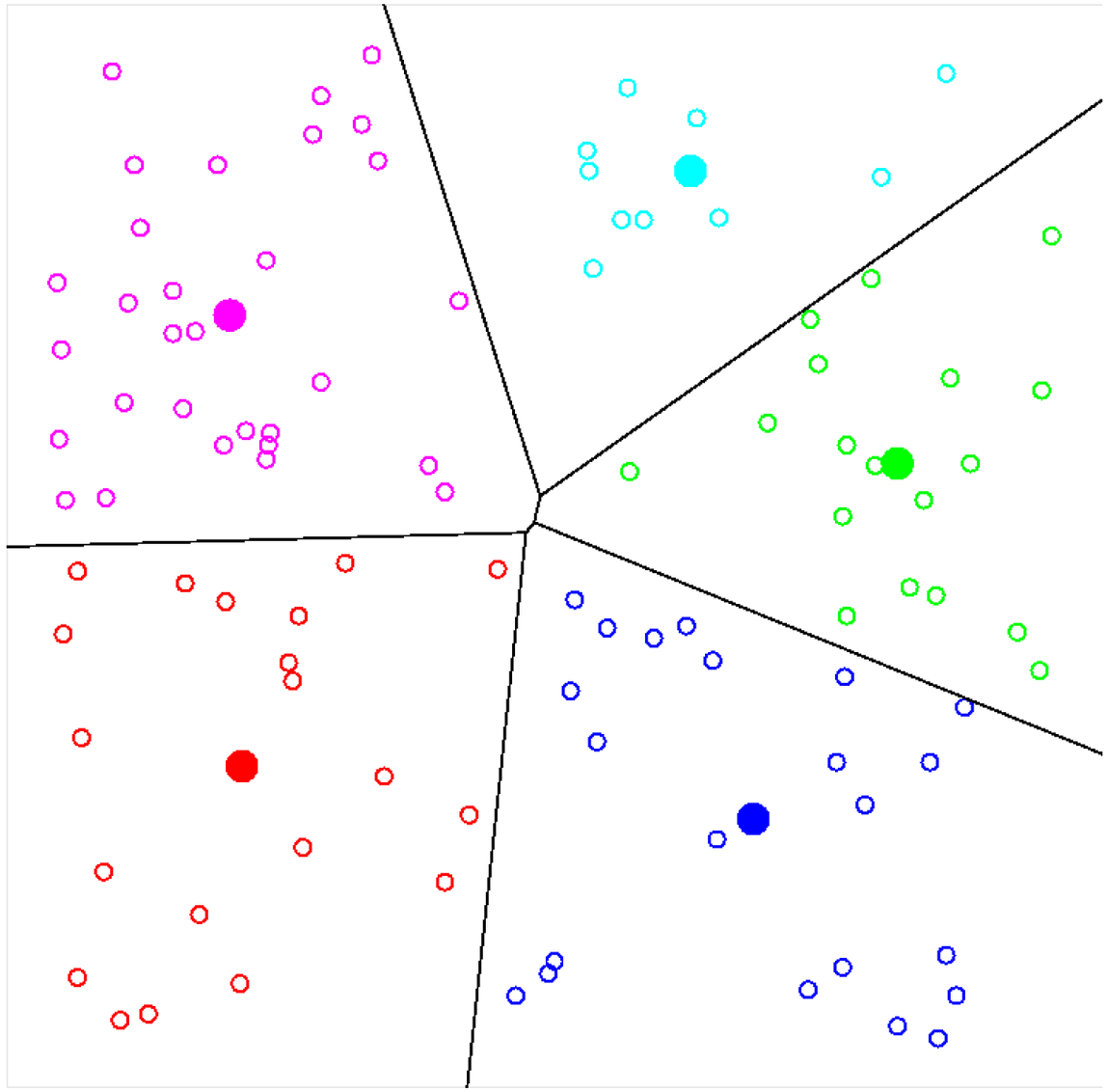
Step 2 Compute a new set of centers/generators from the formula

$$\vec{c}_j = \text{the average of all points in the } j\text{th cluster}$$

Step 3 Check for convergence; if not converged go to Step 1



In this example we have 10 points on $[0, 10] \times [0, 10]$. The initial generators are denoted by red circles and after 3 iterations, they have moved to the black circles. The blue points are in one cluster and the red points in another.



This is our 100 points from a previous slide that have been clustered using K-Means .

Convergence of K-Means

The discrete cluster variance is useful. The standard definition of the discrete variance measures the squared difference between each data item and the mean value, and takes the average.

$$\text{ave(x)} = \frac{1}{n}\sum_{1}^{n} x_i$$

$$\text{var(x)} = \frac{1}{n}\sum_{1}^{n} (x_i - \text{ave(x)})^2$$

This says that if a set of data has a small variance, most of the data is close to the average.

For clustering, we will also want to measure the closeness of data to an average. However, instead of having a single average, we divided our data into clusters, each of which had its own average.

We will need an analogous definition for variance for clusters; the discrete cluster

variance is defined as

$$\frac{1}{n_j}\sum_{x_i \in C_j} \|x_i - \mathrm{ave(x_i)}\|^2 = \frac{1}{n_j}\sum_{x_i \in C_j} \|x_i - c_j\|^2$$

Here $\mathrm{ave(x_i)}$ represents the *cluster average or center* c_j for cluster C_j which contains n_j objects. The total cluster variance would be the sum over all K cluster variances.

Why does K-Means converge?

- Whenever a point is assigned to a new cluster, the sum of the squared distances of each point in the cluster to its assigned cluster center is reduced.
- Whenever a cluster center is moved the sum of the squared distances of the data points from their currently assigned cluster centers is reduced. Thus the cluster variance is reduced.
- If points are no longer moved between clusters then the algorithm has converged.

K-Means should reduce the cluster variance at each iteration but it is NOT guar-

anteed to find the global minimum as the following example illustrates. In this example, with one choice of the starting generators or centers the algorithm terminates at a local minimum and with another choice it finds the global minimum.

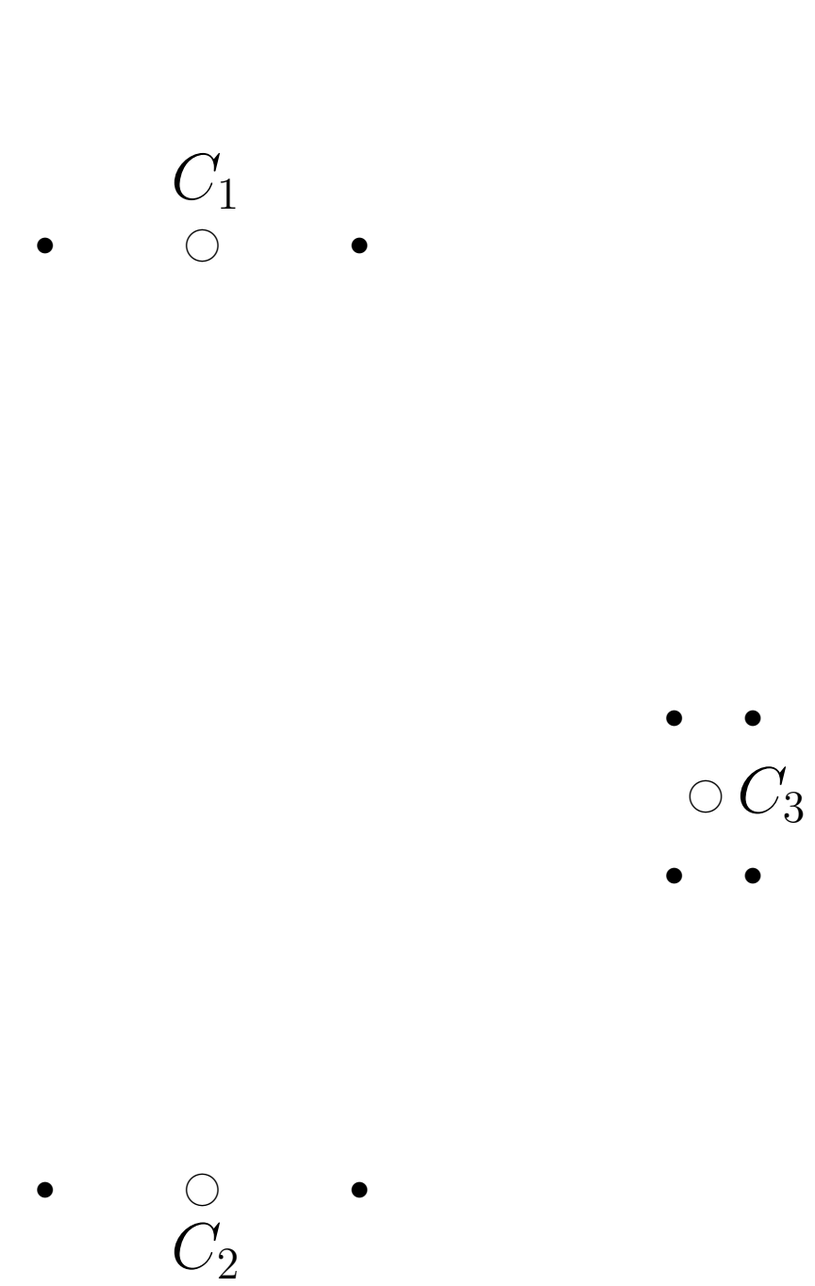
What can we do to prevent K-Means from finding a local minimum?

One approach that is often used is to run the algorithm with different starting points and compute the energies for each and find the global minimum in that way.

Another approach would be to use some information about your data to make a more intelligent choice for the initial generators other than a random choice.

Another approach is to use other quasi-random sampling methods such as Halton sequences, Latin Hypercube, etc.

LOCAL MINIMUM



GLOBAL MINIMUM

Implementing K-Means

Here we will assume that the data to be clustered is finite and of a "reasonable" size.

Initialize: Set k, the number of clusters; input the data to be clustered assuming there are n records and x_i is a generic record; set maximum number of allowable iterations, `max_iters`

Set initial guess for k centers (usually k random records from data set are chosen)

```
for niter = 1:max_iters

    for i = 1:  n
```

find center c_j which record x_i is closest to

increment counter n_j for number of data points in cluster C_j

increment sum of each coordinate of all points in cluster C_j

`end loop over i`

move centers c_j by taking the average of all points in cluster C_j

check for convergence

if converged

compute cluster variance

`break`

`end loop over number of iterations`

To choose the initial clusters you can choose random records in your data set either by scaling the output of `rand` and converting to an integer with `ceil` or `floor` or taking a random permutation of the number of records using `randperm` and taking the first k.

Why do we need to keep a sum of each coordinate of each point in a cluster?

Because to move the centers c_j to their new location we need to take the average of all points in cluster C_j, i.e., use the formula

$$c_j = \frac{1}{n_j} \sum_{x_i \in C_j} x_i$$

How much work does each iteration take?

The main work is in finding the center c_j that the ith record x_i is nearest.

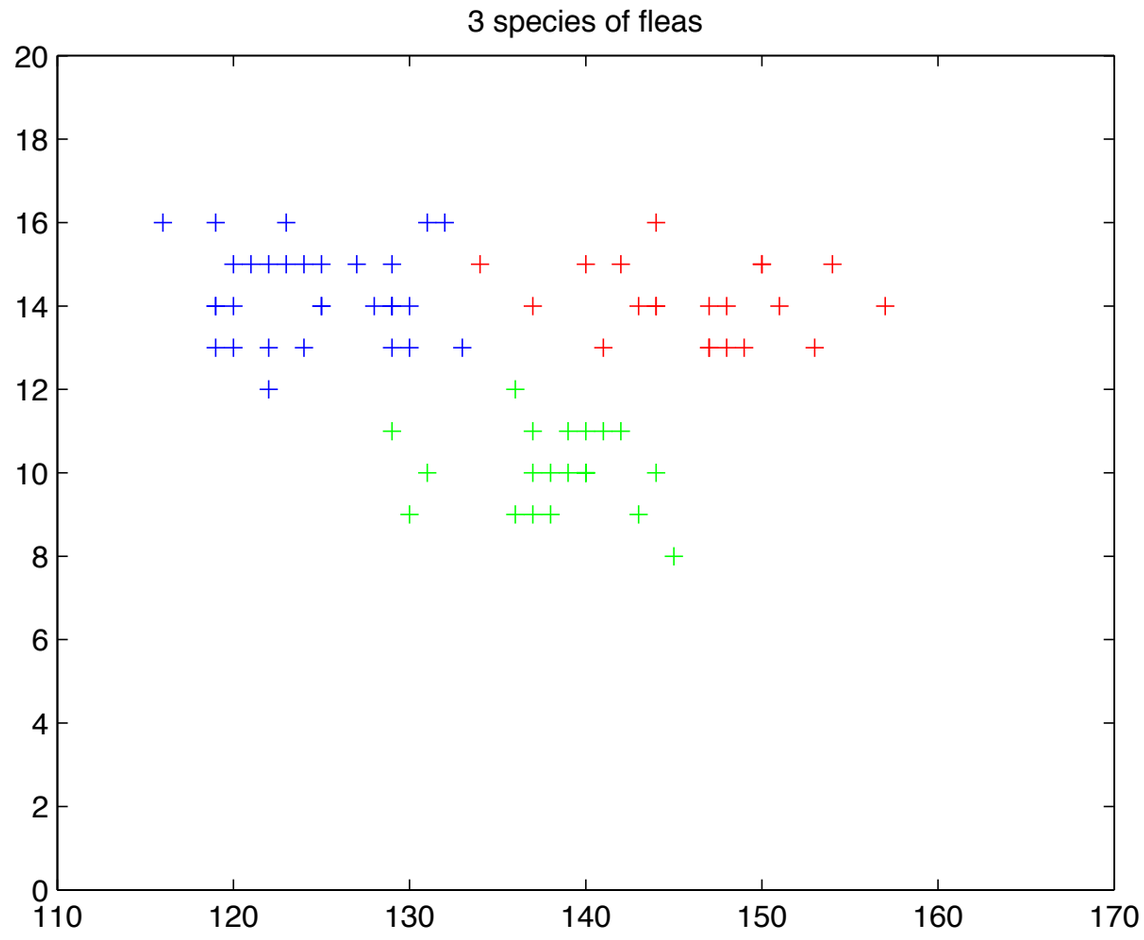
The brute force way to do this is to calculate the distance between x_i and each generator and then take the c_j for which the minimum is attained. The command `[y,loc] =min(·)` is useful because it gives the location of where a minimum occurs. When you look at problems in computational geometry you may investigate more efficient ways to find the closest generator.

How do we decide to terminate the iteration? One way is to check that the

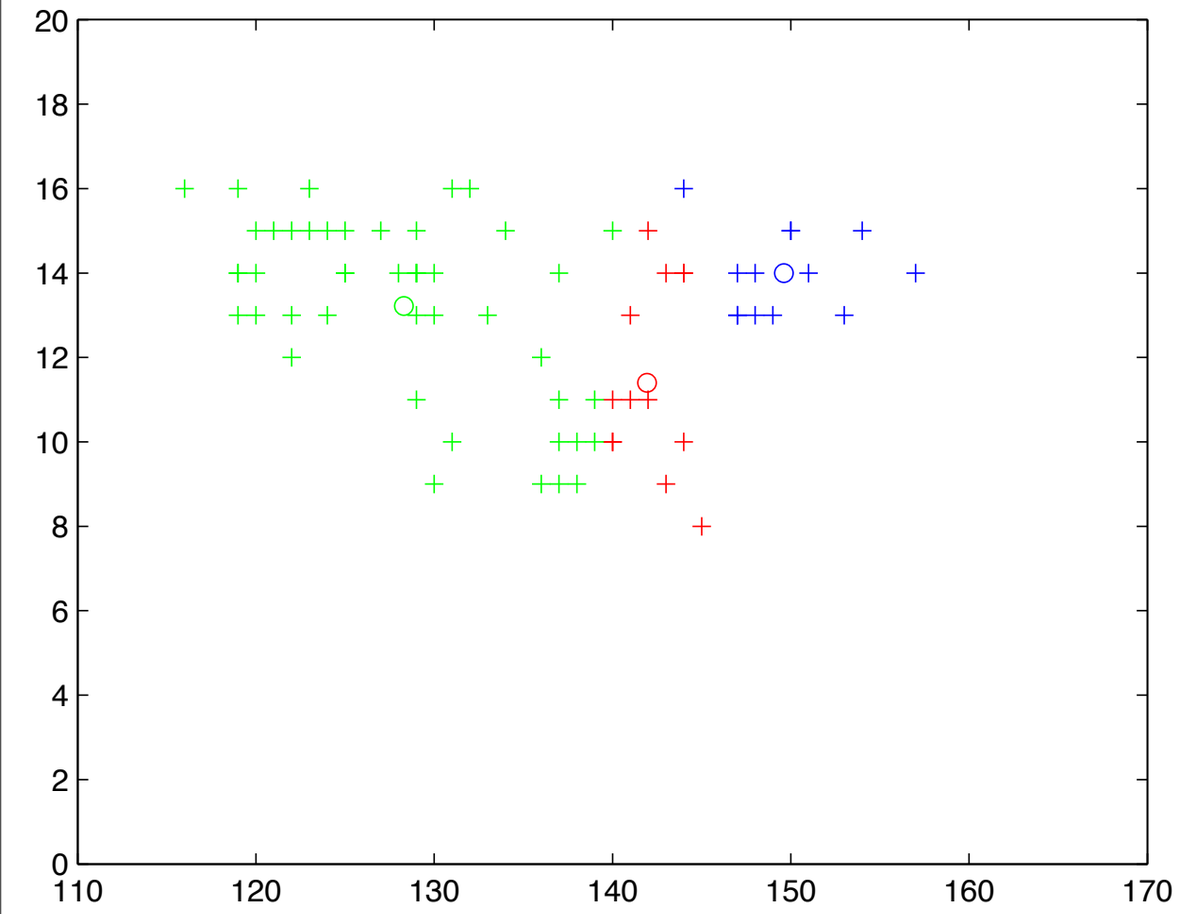
cluster centers are no longer moving very much. For example, if c_j^k denotes the jth center at the kth iteration then we could check

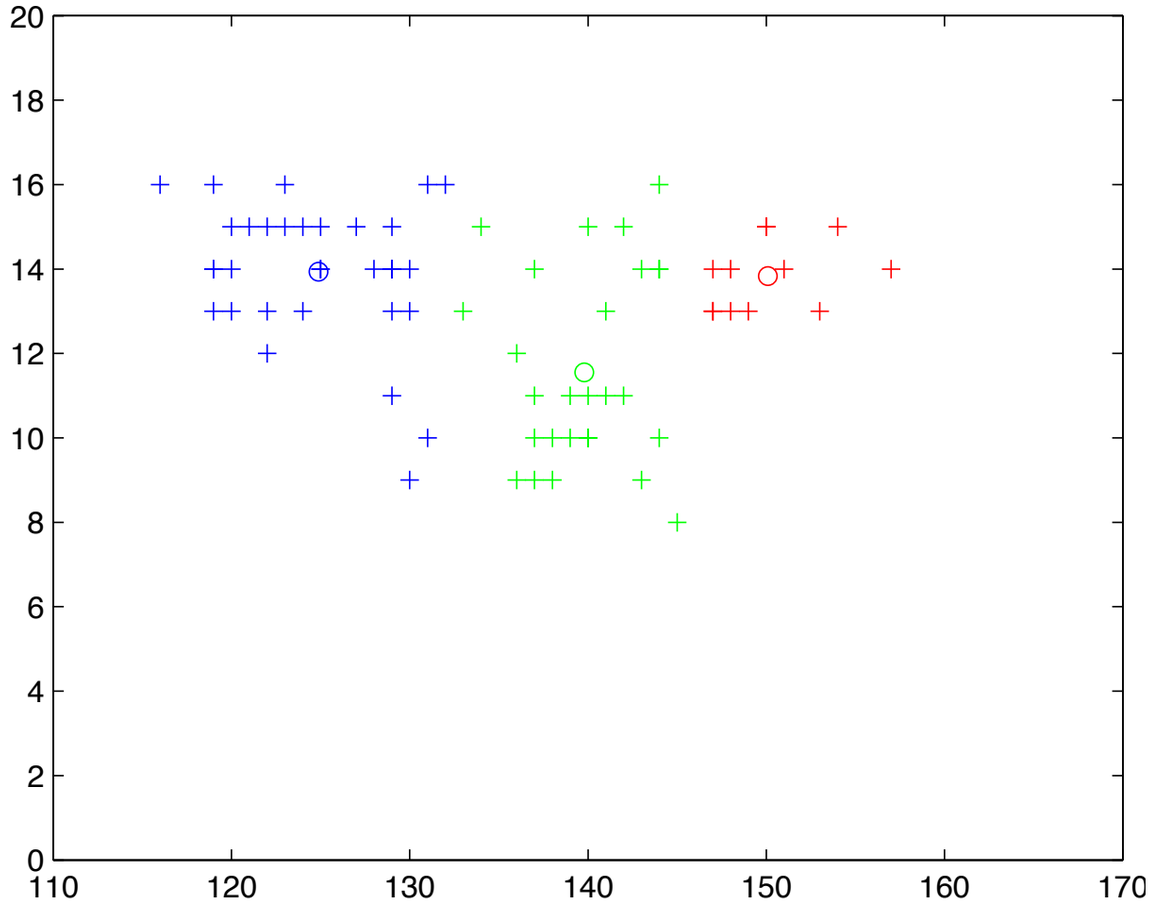
$$\max_{j=1,\ldots,k} \|c_j^{k+1} - c_j^k\| \leq \text{tolerance}$$

Example Use K-Means to cluster the set of data for 3 species of fleas shown in the figure below.

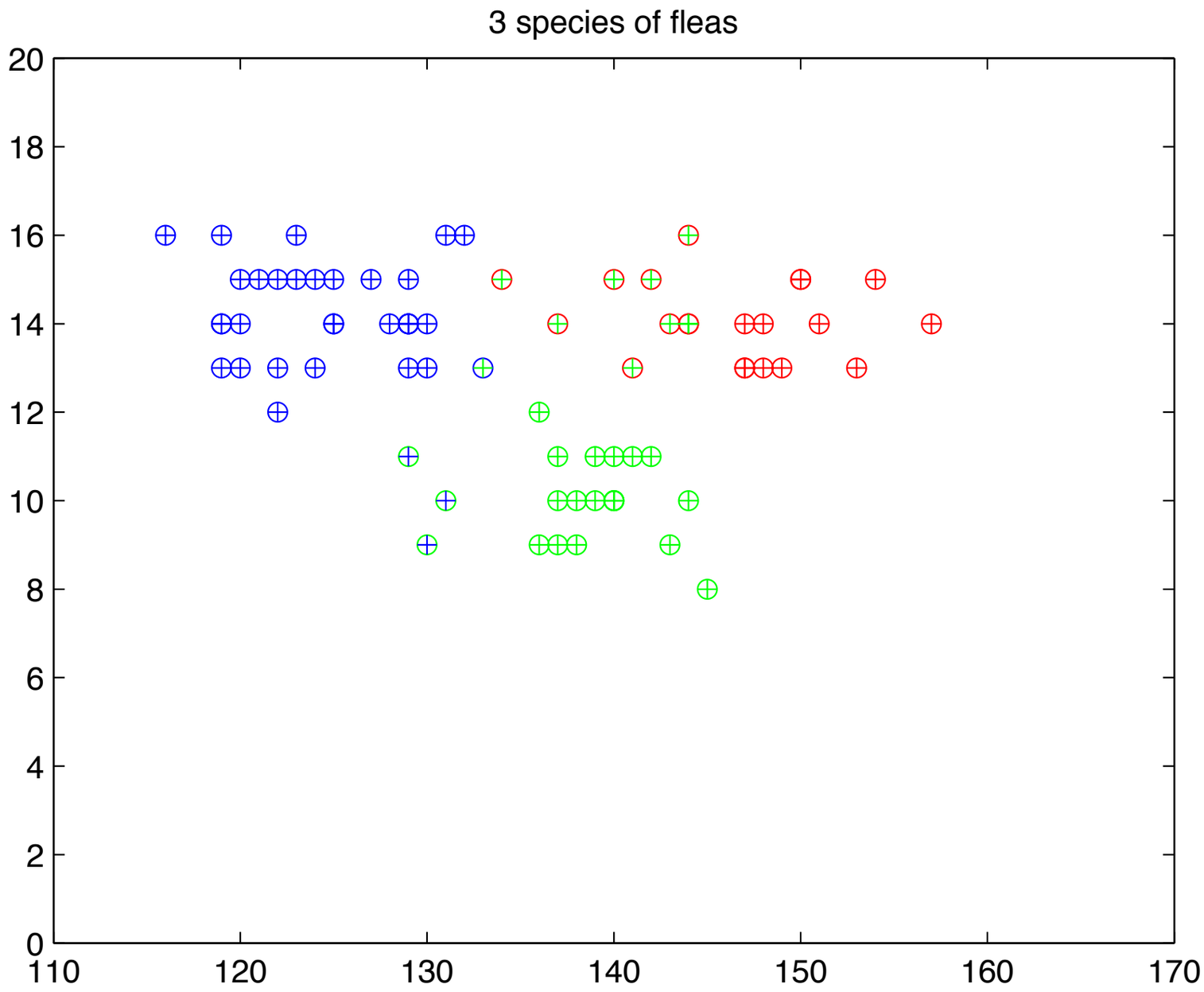


The figure on the left below shows the clusters after the first iteration and the figure on the right gives the final clusters.



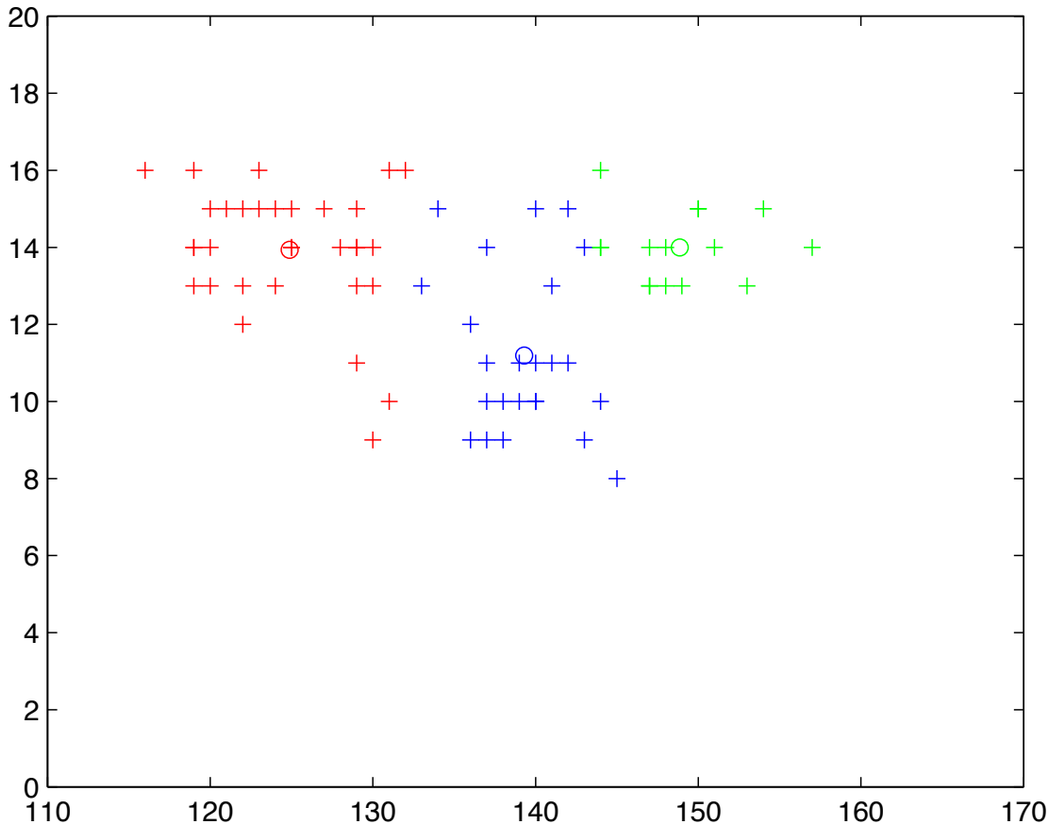


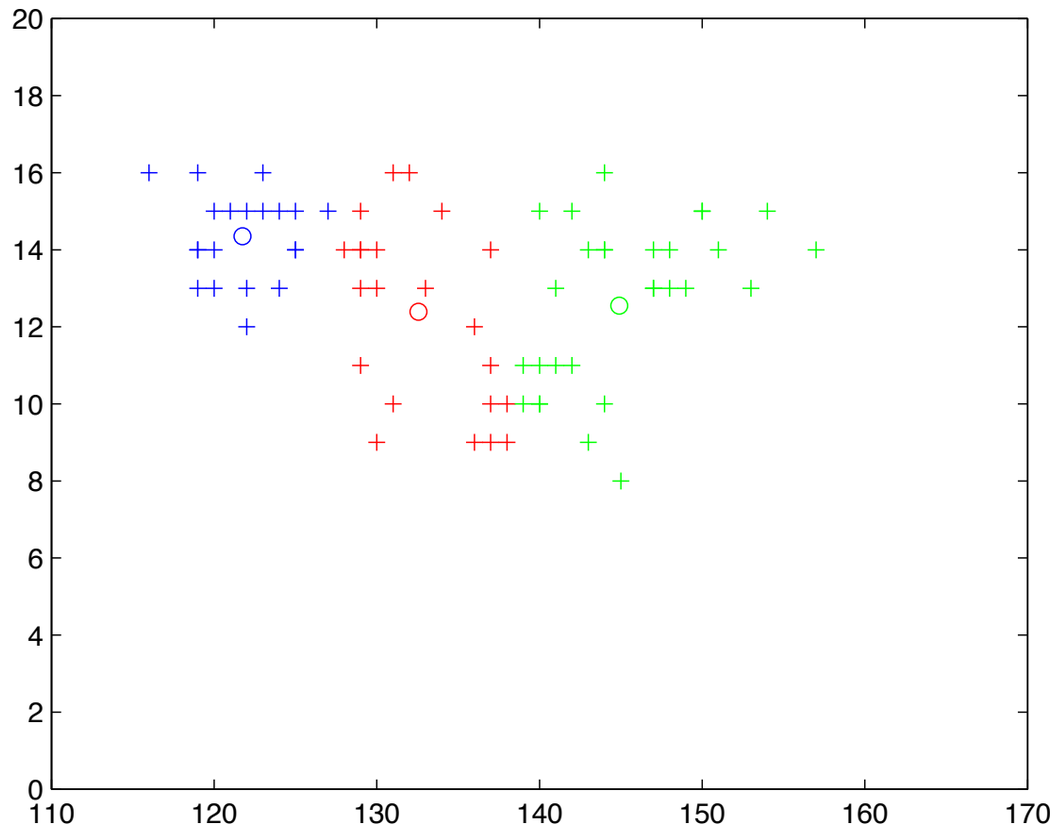
Here is a comparison between the natural clusters and our result from K-Means. The circle indicates the natural clusters and the plus sign the clusters we have obtained.



Both of the figures below produce results where K-Means converged to a given tolerance.

Why do we get different results?





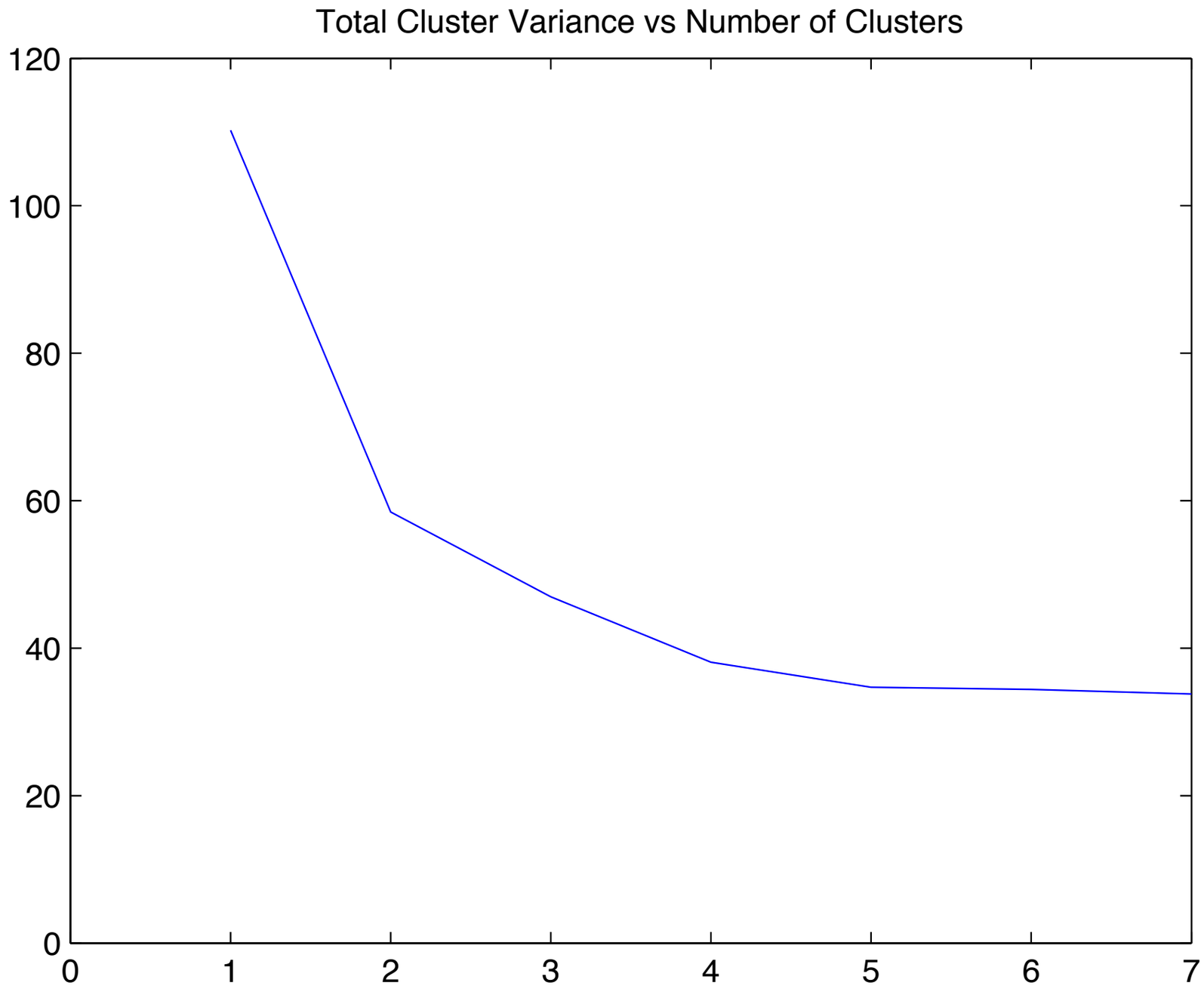
K-Means is sensitive to the initial guess. Here we are choosing random initial guesses and because the data is relatively small (only 74 records) we could easily choose 2 or even all of the initial centers from one of the classes of data.

How do we know which clustering to accept?

Oftentimes one runs the code for several choices of initial centers and accept the one which gives the smallest total cluster variance.

What if we didn't know that the data was naturally clustered into 3 groups?

We could run the code for different numbers of clusters and see what happens. In our example we get the following cluster variances. We report the smallest cluster variance from several choices of initial clusters.



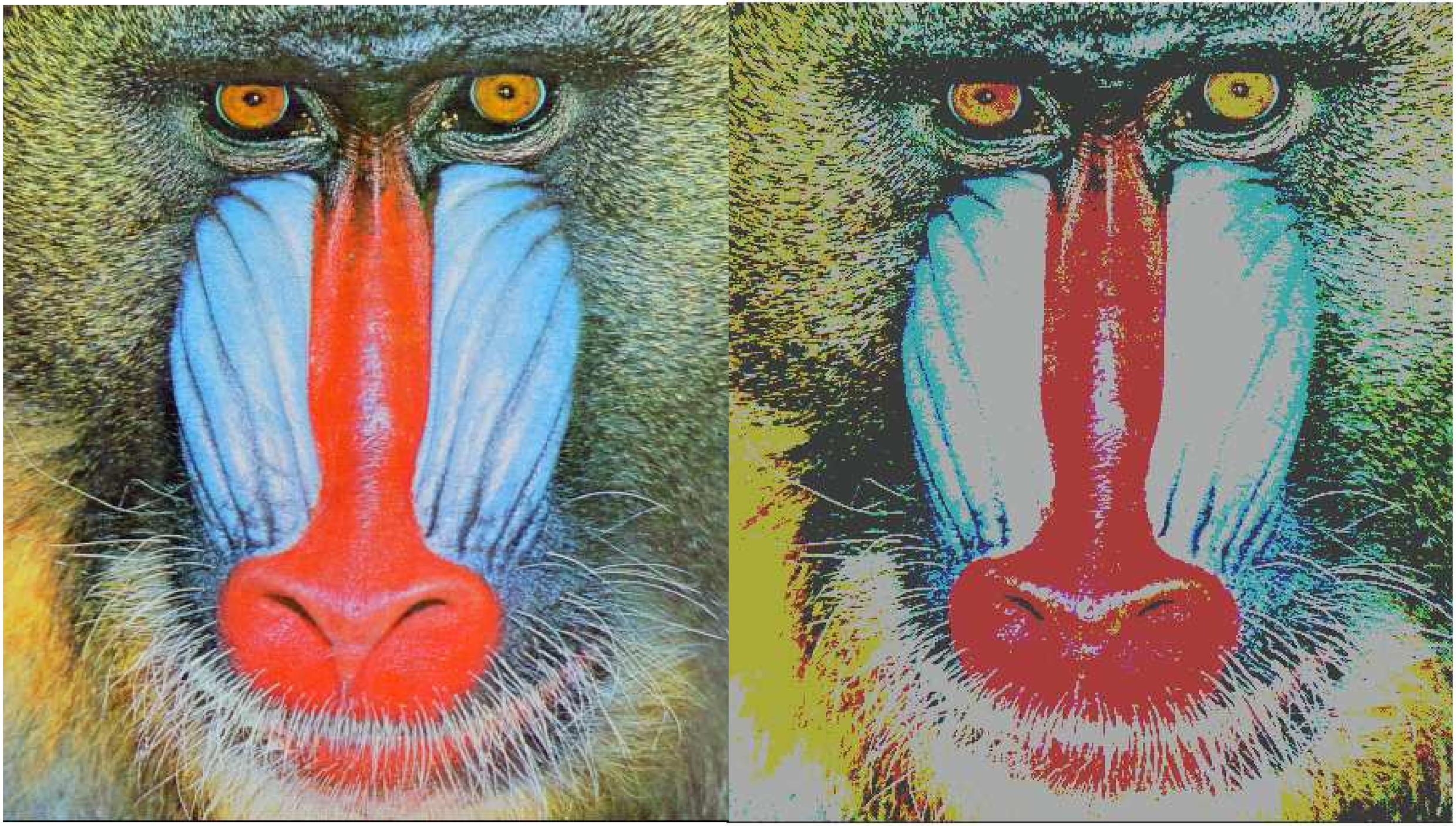
This plot indicates that the natural number of clusters is either 3 or 4.

Using K-Means for Image Compression

- If we have a color image we know that each pixel is represented by three RGB values creating a myriad of colors.
- Our your computer monitor you have essentially an unlimited number of colors available but on a printer you have less.

  For example, suppose you only had 32 grayscale colors available on a printer and you wanted to determine print an image with many more. If a pixel is represented by a shade of gray other than the 32, how can you assign a shad to that pixel?
- We can use K-Means to accomplish this image compression. In the lab we will find which 32 colors best represent the image.
- To do this, we initiate our probabilistic Lloyd's algorithm with 32 generators which are numbers between 0 and 255; we can simply choose the generators randomly.

- In Lloyd's algorithm we need to sample each record so in our application this means to sample the image; i.e., sample a random pixel. If the image is not too large, then we can simply sample every pixel in the image.
- We then proceed with the algorithm until convergence is attained.
- After convergence is achieved we know the best 32 shades of gray to represent our image so our final step is to replace each color in our original matrix representation of the image with the converged centroid of the cluster it is in.



The image on the left is the original image and the one on the right uses 8 colors to represent it.

Using random pixels for an image

```
Pick x pixel randomly from a picture with n pixel using x random coordinates x,y

Run k-means clustering on the x pixel finding m new centers c

For i=1 to n:
      for j=1 to m:
        calculate distance(p(i),c(j))
        record z=j with lowest distance
      pixel(i) = c(z)

 Print picture
```

Weighted K-Means Clustering

- We now consider some extensions of the K-Means algorithm which let us consider new classes of problems.
- If we know that some data has greater importance, we would like to make sure that the information is included. Consequently we need a mechanism for the clustering to take into account the relative importance of the information.
- Suppose that we want to cluster a set of data points X, but now we want to be able to stipulate that some points are more "important" than others.
- We might encounter this problem if we are deciding where to move the capital city of a state, or put a new hub for an airline

  Here we are trying to choose convenient "centers", but now it's not just the geographic location that we have to consider, but also $how\ many$ people or average flights will have to travel from the data points to the cluster centers.

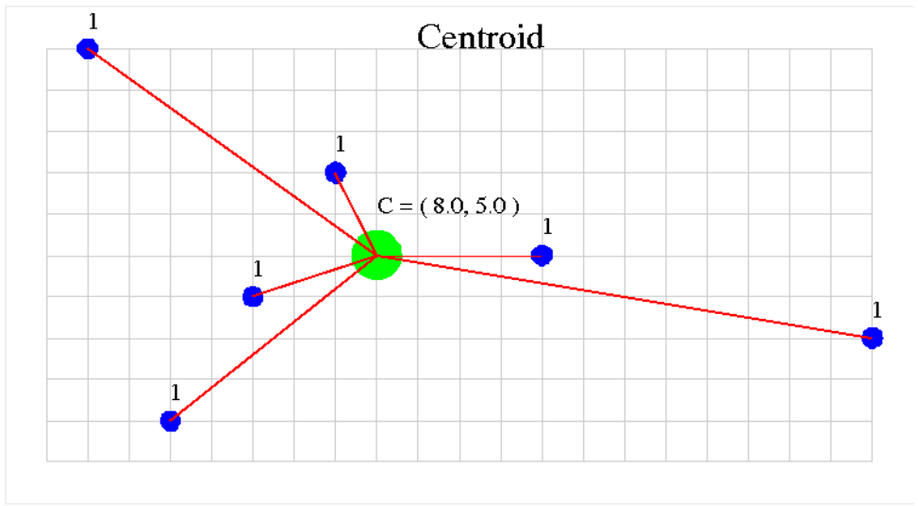
The Centroid

- Recall that in the unweighted case, a cluster center c_j was the centroid or average of the coordinates of all the data points x_i in the cluster C_j:

$$c_j = \frac{\sum_{x_i \in C_j} x_i}{\sum_{x_i \in C_j} 1} = \frac{1}{n_{C_j}} \sum_{x_i \in C_j} x_i$$

  where n_{C_j} is the number of points in cluster C_j

- The centroid is a geometric quantity whose location can be roughly determined by sight. If we imagine the points being connected to the centroid, and having equal weight, then this object is perfectly balanced around the centroid. No matter how we turn it, it will be balanced.

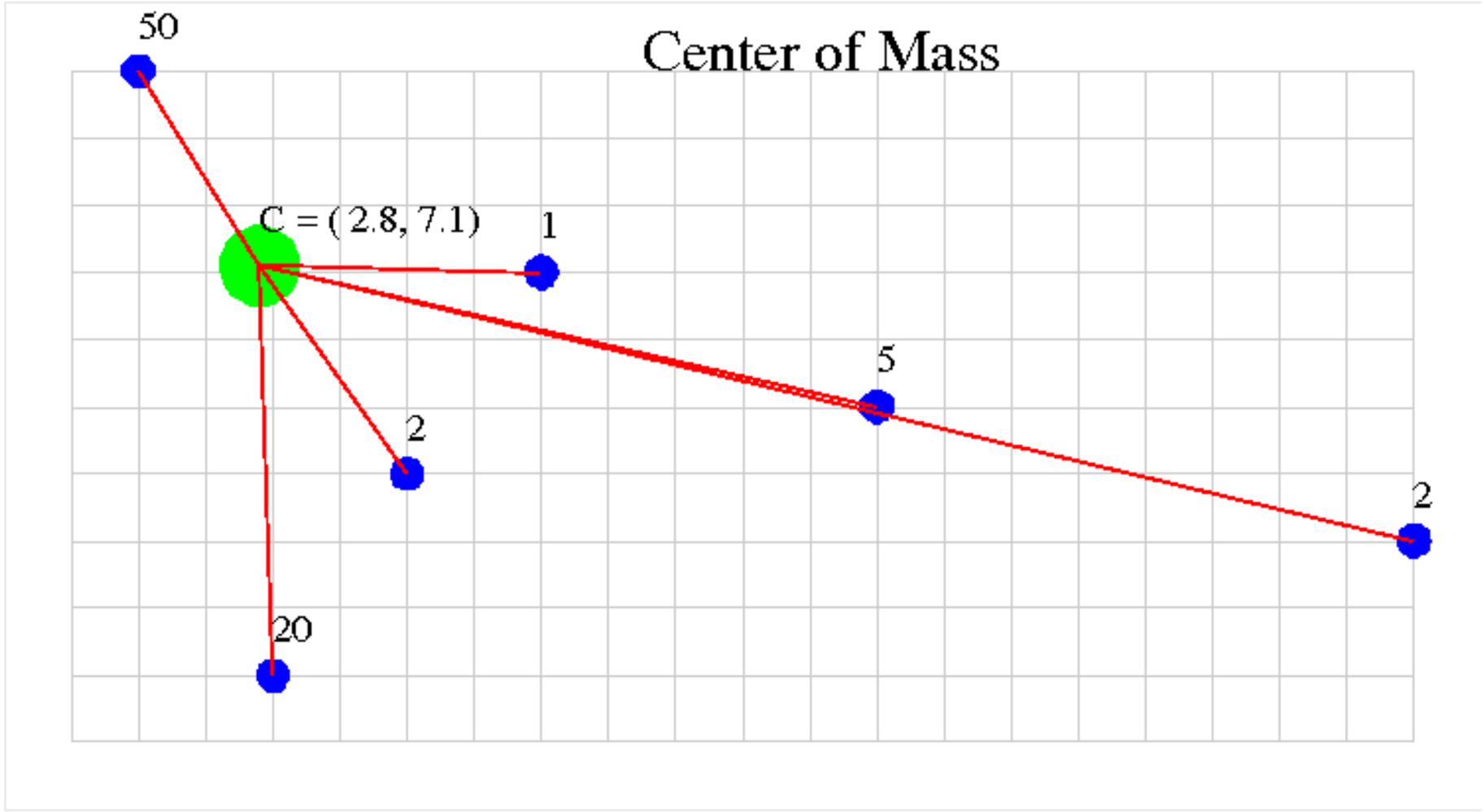


Center of Mass

- When each point has a different weight or importance then we use the center of mass.
- Assume point x_i has weight w_i. Then the formula for the discrete center of mass is:

$$\mathrm{c_j} = \frac{\sum_{x_i \in C_j} w_i \, x_i}{\sum_{x_i \in C_j} w_i}$$

  and cluster center c_j is the center of mass of cluster C_j.
- The location of the center of mass can be $anywhere$ within the convex hull of the set of data points. (The convex hull is smallest convex set containing all points; to visualize it, consider the points as nails in a board and stretch a rubber band around the outside of the nails – this is the convex hull.)



If each point x_i is connected to the center of mass, and given weight w_i, this object will also be perfectly balanced.

For weighted clustering, we define the weighted cluster variance as:

$$\sum_{j=1}^{K} \mathrm{var}(\mathrm{x}, \mathrm{w}, \mathrm{c_j})$$

$$= \sum_{j=1}^{K} \frac{\sum_{x_i \in C_j} w_i \|x_i - c_j\|^2}{\sum_{x_i \in C_j} w_i}$$

where c_j is the cluster center.

Each step of the weighted K-Means algorithm reduces the weighted cluster variance, and the best clustering minimizes this quantity. As in the case of unweighted K-Means , the algorithm can get stuck at a local minimum.

Voronoi Diagrams - An Early Example

The spread of cholera in England presents an early example of the use of a Voronoi Diagram.

In the early nineteenth century, European doctors working in India reported a strange new disease called cholera which almost always resulted in an agonizing death.

It was reported that the disease would start in one village, kill most victims within three days and would break out in neighboring villages.

Doctors at that time believed that cholera was transmitted by miasm: an invisible evil-smelling cloud of disease. Miasm explained why cholera victims often lived in poor areas full of tanneries, butcher shops, and general dirty conditions.

Cholera spread to England and Dr. John Snow became interested in the patterns

of its spread and realized that the theory of miasm didn't fit the pattern.

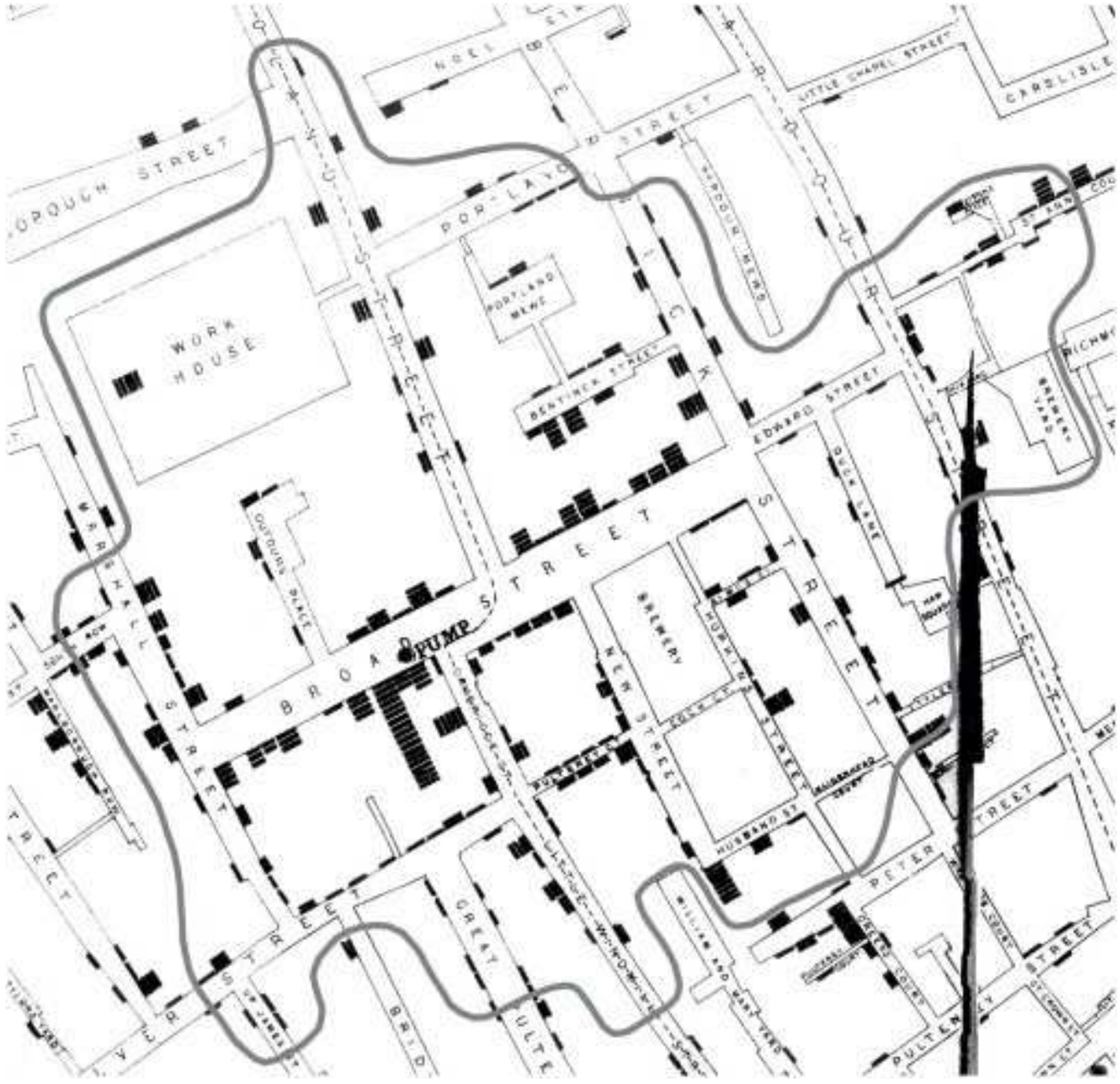
In one small outbreak of cholera, only people living near the Thames river got sick. To Dr Snow, this suggested something in the river water. A second outbreak occurred away from the river, in an area with piped-in water. People getting water from one company were healthy, while others got sick. The intake pipes for that one company were located upstream from London.

At that time in London there was no running water . For drinking, cooking, cleaning and bathing, they went to one of the town pumps which drew water directly from the ground. Most people had no sewage system . People used chamberpots and buckets, which were emptied into cisterns and carted off by an informal network of night-soil carters.

Dr. John Snow suspected the water pump on Broad Street, but he needed evidence (no one knew about germs):

- He made a map of the district.
- He marked every house where cholera victims had lived;

- He paced the distance to the nearest pumps;
- The houses closest to the Broad Street pump were circled by a black line.



The area around the Broad Street water pump; all houses which were closest to this pump are in the group and homes where cholera victims lived are marked.

Snow's map strongly suggested that people who died were precisely those for whom the Golden Square pump was the closest.

He personally interviewed people to explain exceptions to the pattern he found such as healthy workers at a beer factory who didn't drink any water; some sick children lived outside the district, but came to school there; a woman who lived miles away had her son deliver "fresh" water from the Broad Street pump because she liked its taste.

Dr. Snow's map destroyed the miasm theory, because it could show where deaths occurred in a way that suggested why.

Dr. Snow was doing epidemiology, but also mathematics. His map is an interesting example of a Voronoi diagram.

A couple of books have been written about Dr. Snow and his work with cholera.

"The Strange Case of the Broad Street Pump" by Sandra Hempel.
"The Ghost Map" by Steven Johnson

Voronoi Diagrams

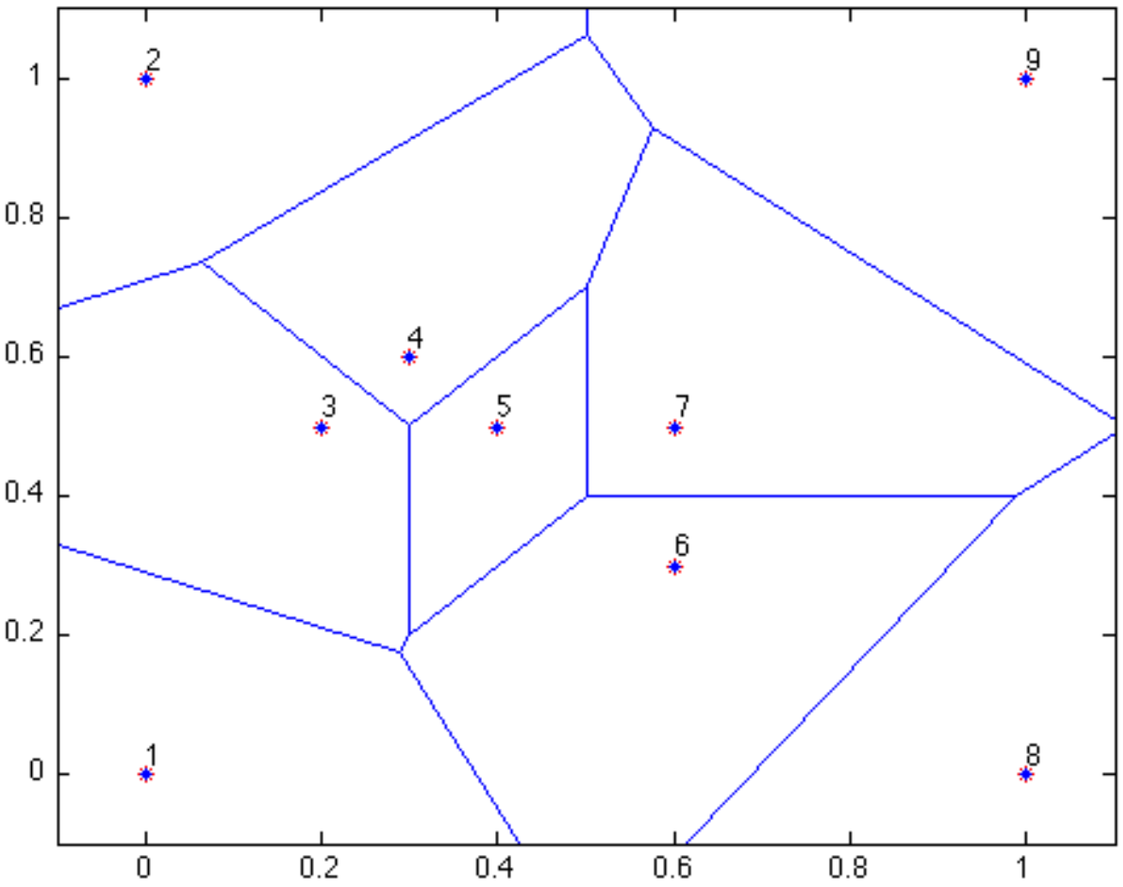
We can view a Voronoi Diagram as a means of clustering when the data is discrete.

Suppose that

- we have a set $\mathcal{S}$ of data which can be finite or infinite
- we have a set $\mathcal{C}$ consisting of objects z_i called generators (not necessarily in $\mathcal{S}$);
- we have a distance function $d(x_i, z_j)$ for $x_i \in \mathcal{S}$ and $z_j \in \mathcal{C}$.

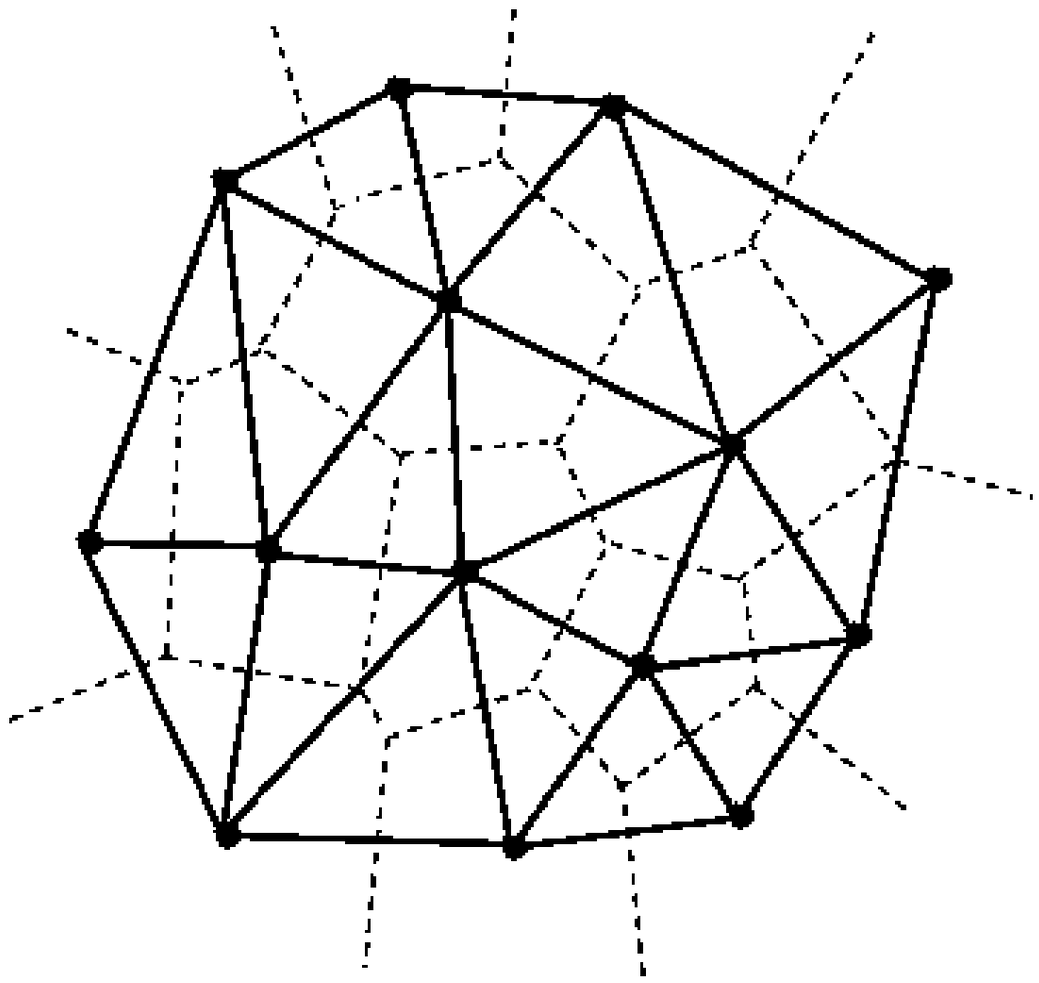
The Voronoi set $\mathcal{V}_j \subset \mathcal{S}$ consists of all $x_i \in \mathcal{S}$ which are closer to z_j than to any other generator.

The collection of all subsets $\{\mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_K\}$ is called a Voronoi tessellation or Voronoi diagram of $\mathcal{S}$.



Note that the Voronoi regions for two points in the plane are the regions on either side of the perpendicular bisector to the line joining the points.

With each Voronoi tessellation there is an associated Delauney triangulation which is found by connecting the generator and its two nearest generators. This is useful for grid generation.

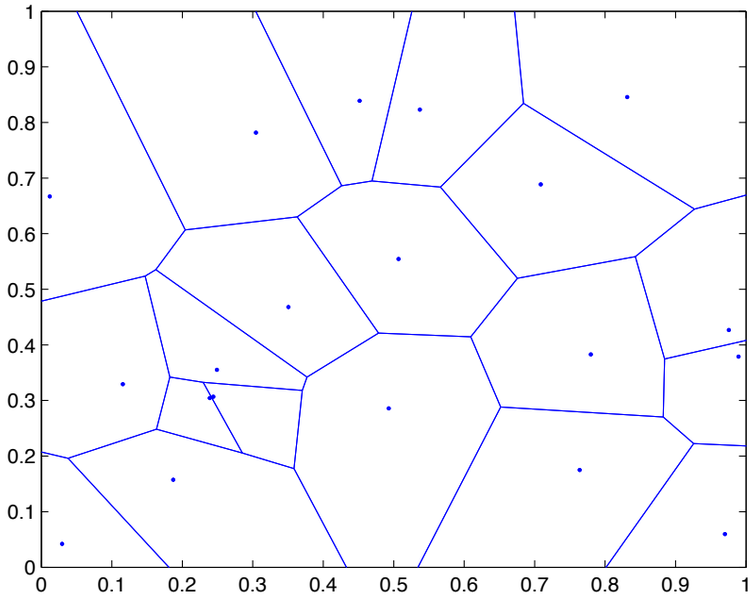


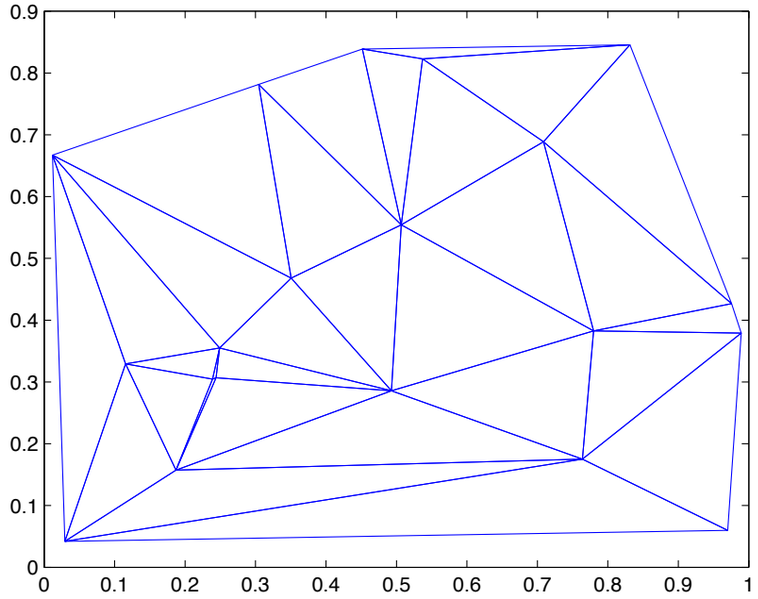
The dotted line represents the Voronoi Tessellation and the solid lines the Delauney triangulation.

Matlab has a command for generating a Voronoi Tessellation of a two dimensional region and commands for the associated Delauney triangulartion.

Example Use Matlab to generate 20 random points in $\mathbb{R}^2$ and draw the Voronoi tessellation. Use the same points to draw the associated Delauney.

```
x= rand(20,1);

y = rand(20,1);

voronoi(x,y) % draws Voronoi diagram

dt = DelaunayTri(x,y);% computes Delaunay

triplot(dt) % draws Delaunay triangulation
```


0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Calculating Voronoi regions by sampling

Suppose you have a region in $\mathbb{R}^2$ and you want to write an algorithm to obtain a Voronoi tessellation.

In this case we want to determine all points which are closer to generator z_i than to any other generator z_j. However, there are an infinite number of points.

In this case we don't have a finite number of points to cluster so instead we "sample" the region. If we choose enough sampling points then we will get a reasonable Voronoi diagram.

For example, if we want to create a color Voronoi diagram in the unit square with $M \times N$ pixels using a sampling approach then we could perform the following steps.

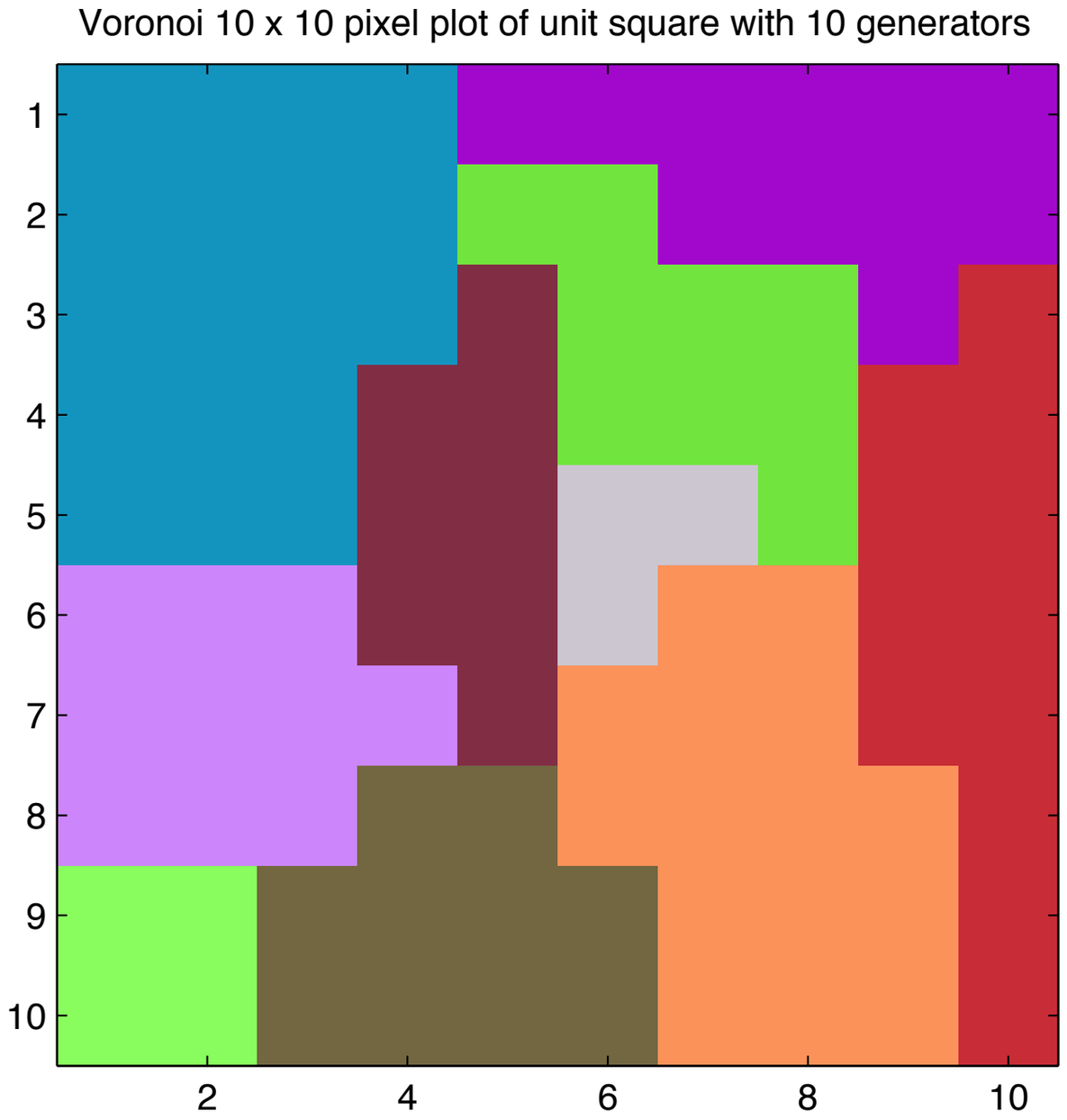
- choose $\mathbf{K}$ distinct colors $\mathbf{RGB(1{:}K)}$, one for each center $\mathbf{C}$;

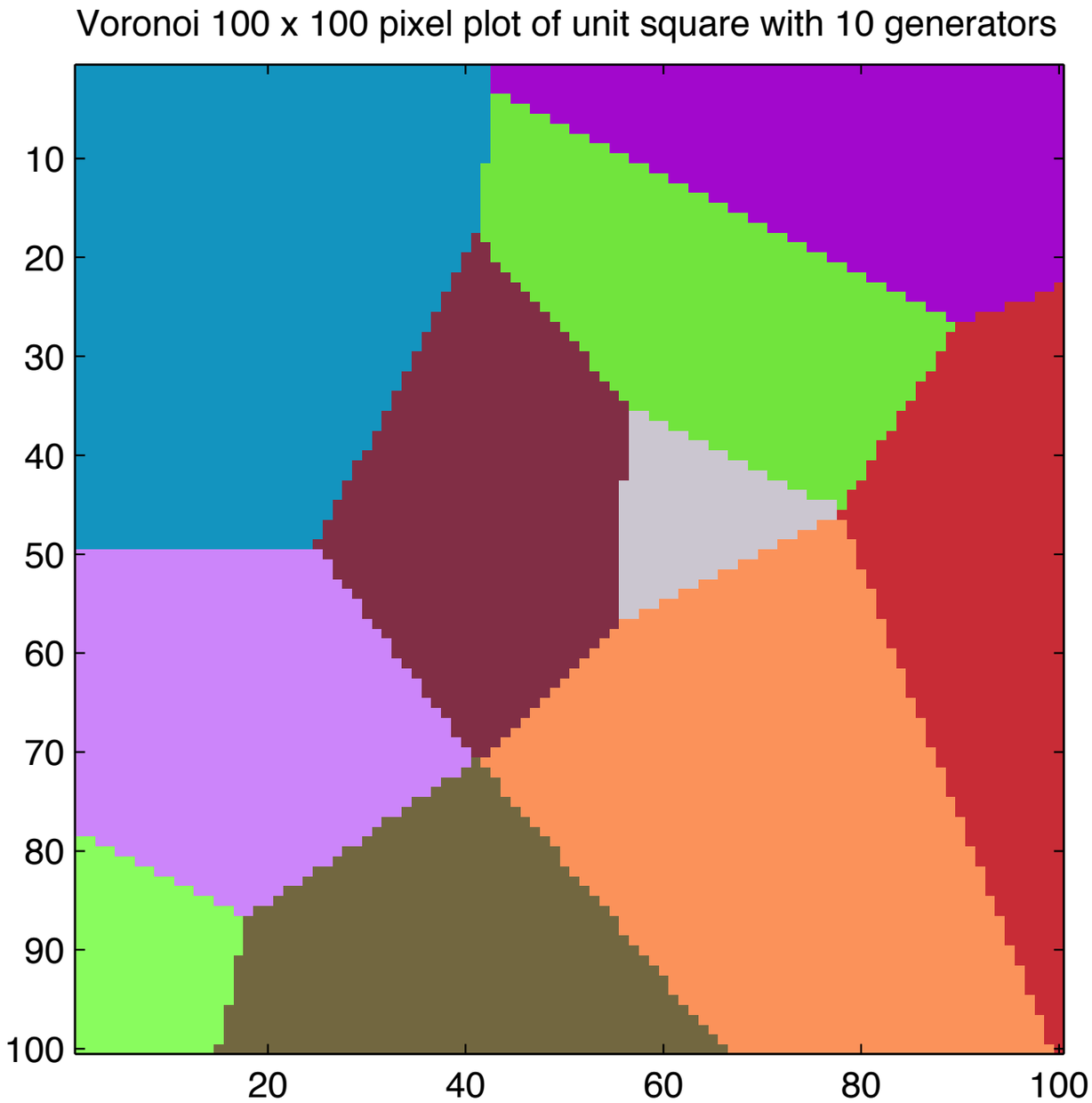
- use the map (I,J) $\rightarrow$ (X,Y) = ((J-1)/(N-1),(M-I)/(M-1));
- for every pixel (I,J), compute the (X,Y) coordinates;
- find $\mathbf{C^*}$, the center which is closest to (X,Y);
- set $\mathbf{A(I,J) = RGB(C^*)}$.

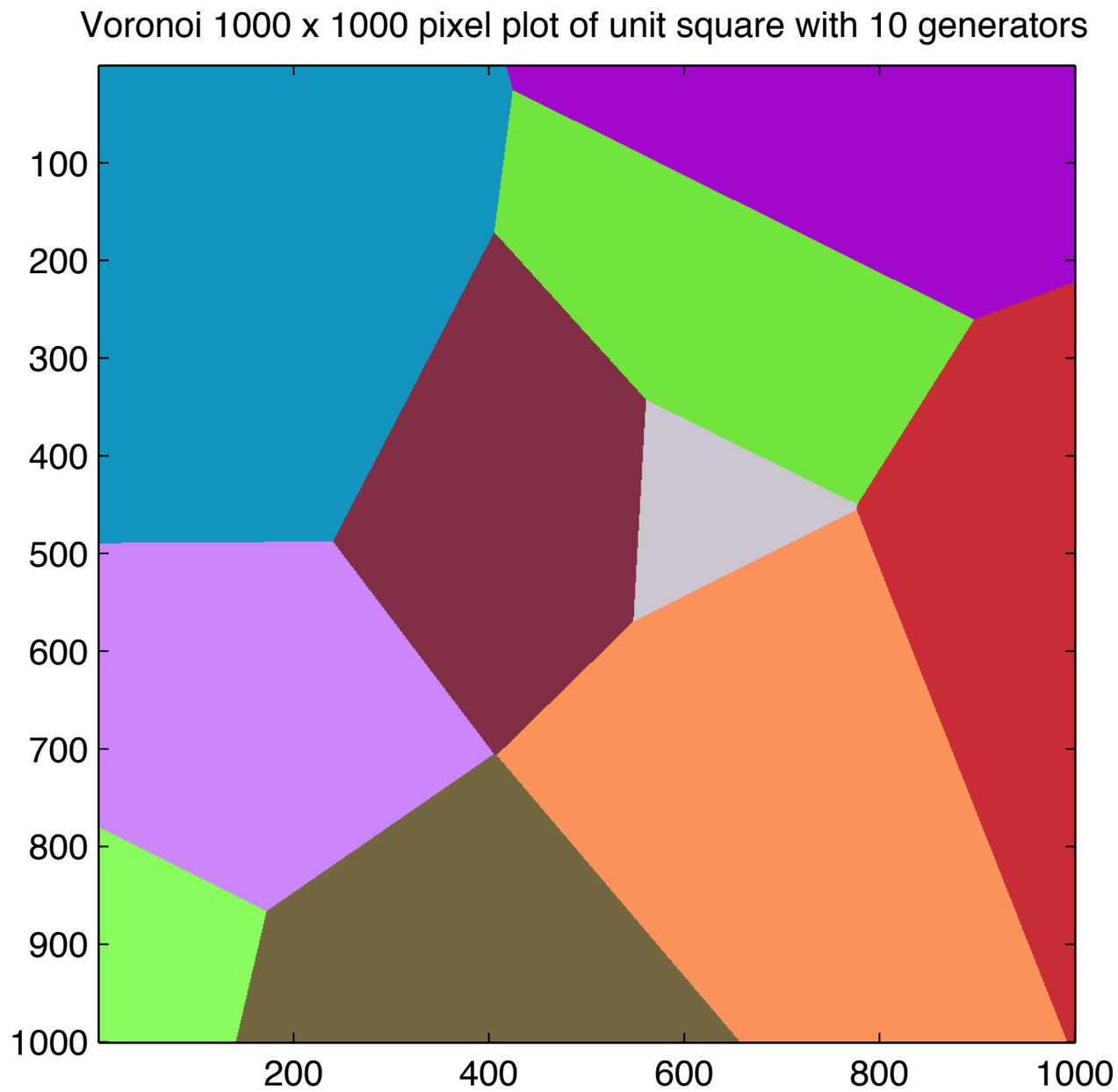
```
% Input M, N, NC (# clusters)
%  Randomly choose NC + 1 sets of RGB values.
%  Our extra color is black, just in case something goes wrong.
%
  rgb = uint8 ( floor ( 256 * rand ( nc + 1, 3 ) ) );

  rgb(nc+1,1:3) = 0;
%
%  For each pixel in A, we calculate its correspoinding XY position,
%  find the nearest center, and color the pixel with the correspondi
%  RGB color.  A vectorized calculation would be much faster.
%
%  The L2 norm is used here.
%
  a = uint8 ( zeros ( m, n, 3 ) );

  for i = 1 : m

    y = ( ( m - i ) * ymax  + ( i - 1 ) * ymin )  / ( m - 1 );
```

```
for j = 1 : n

  x = ( ( n - j ) * xmax  + ( j - 1 ) * xmin )   / ( n - 1 );

  nearest = nc + 1;
  distsq_min = Inf;

  for k = 1 : nc

    distsq = ( x - xy(1,k) )^2 + ( y - xy(2,k) )^2;

    if ( distsq < distsq_min )
      distsq_min = distsq;
      nearest = k;
  end
end

a(i,j,1:3) = rgb(nearest,1:3);
```

```
    end

  end
%
%  Display the image.
%
  imagesc ( a )
```

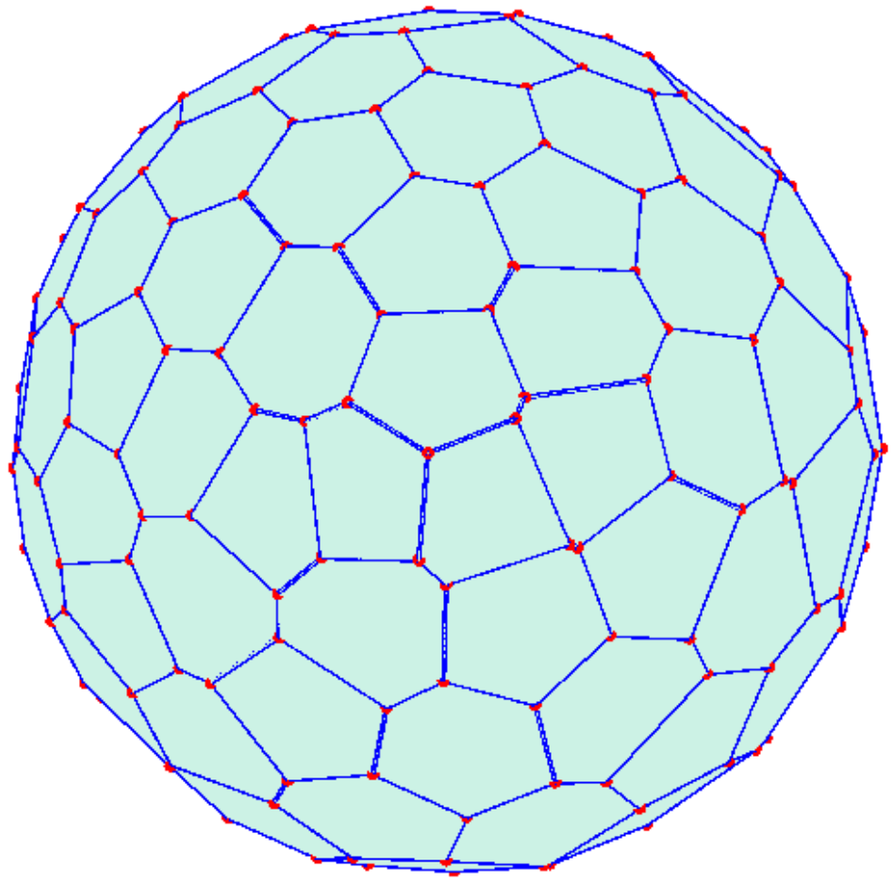

Voronoi 10 x 10 pixel plot of unit square with 10 generators
1
2
3
4
5
6
7
8
9
10
2
4
6
8
10


Voronoi 100 x 100 pixel plot of unit square with 10 generators
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100

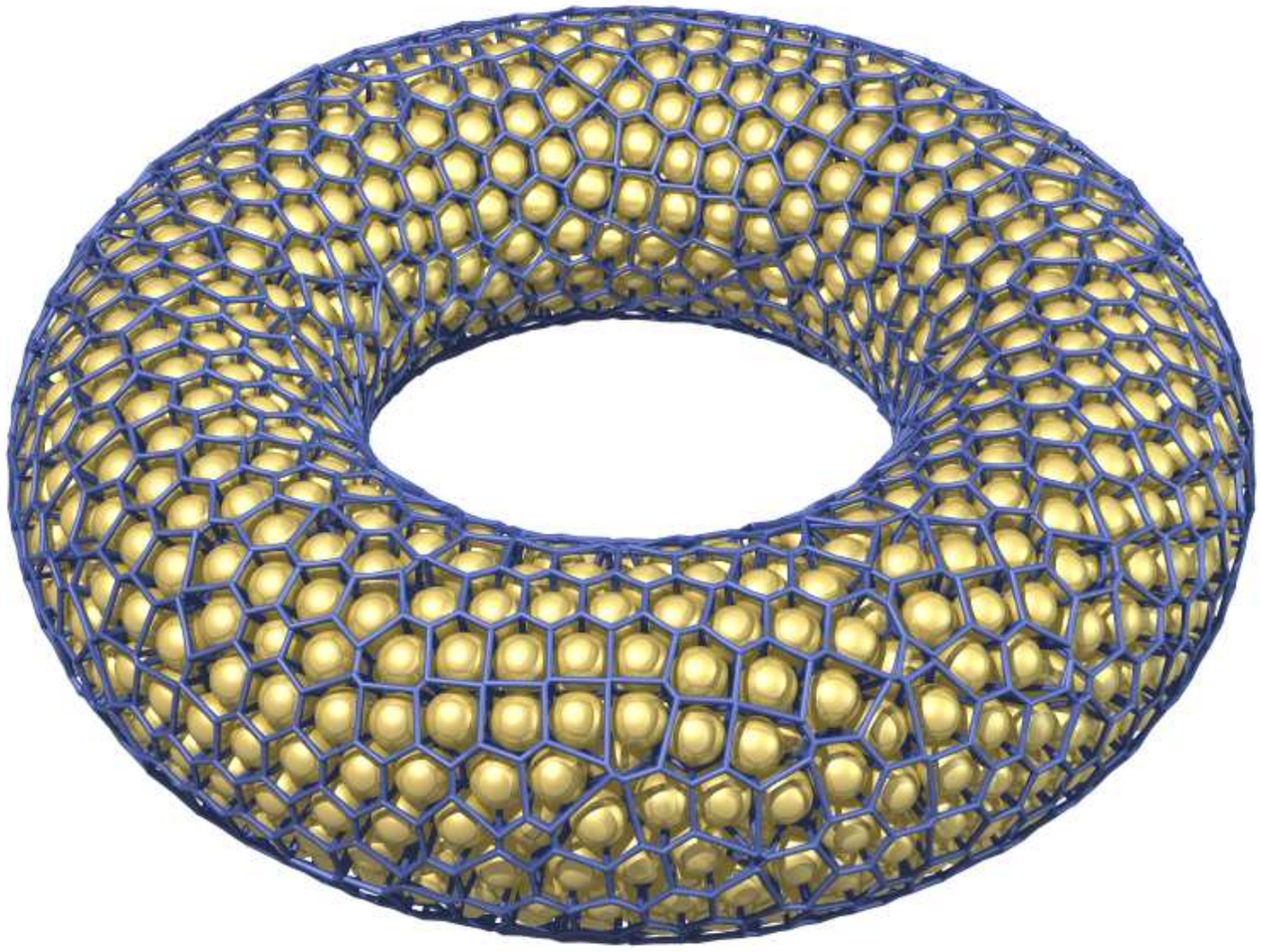

Voronoi 1000 x 1000 pixel plot of unit square with 10 generators
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000

If we want a Voronoi diagram on a more complicated domain, then it is advantageous to use existing software.

A Voronoi tessellation on the surface of a sphere (for grid generation in climate modeling) generated using Stripack



A Voronoi tessellation on the surface of a torus generated using Voro++



Centroidal Voronoi Tessellations

As before, we can define a center of mass z^* for each $\mathcal{V}_i$. If we have continuous data then the formula is

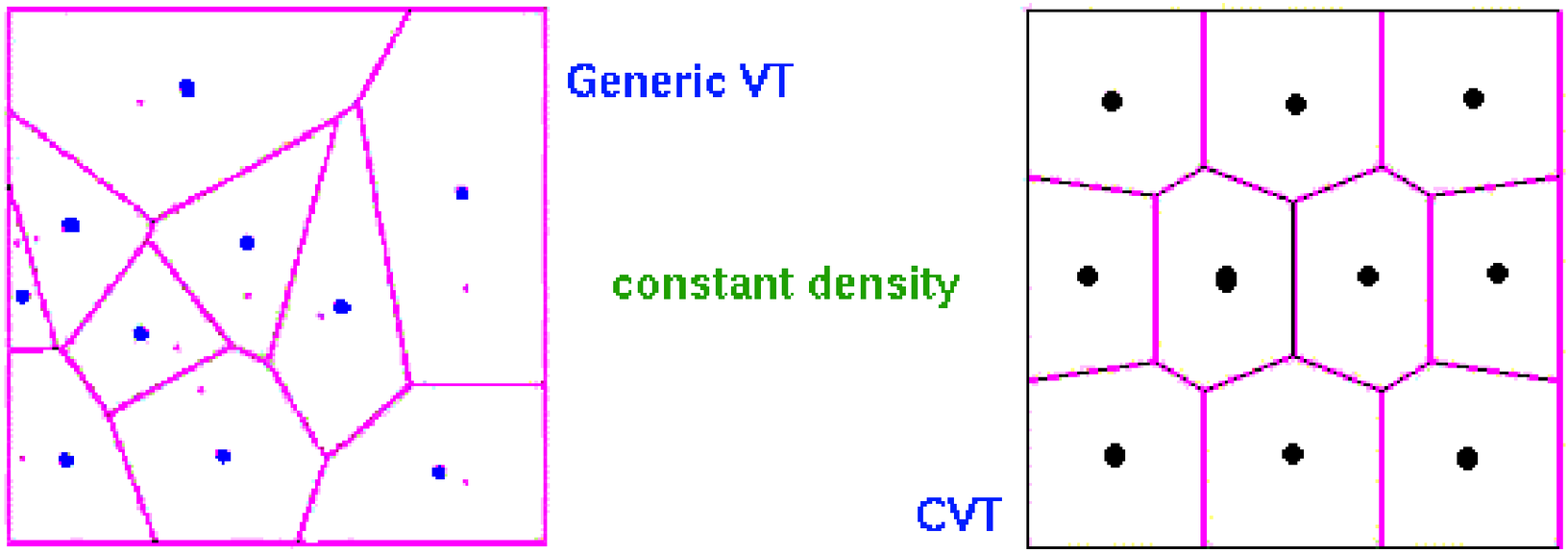
$$z^* = \frac{\int_{\mathcal{V}_i} \rho(w) w \, dw}{\int_{\mathcal{V}_i} \rho(w) \, dw}$$

where ρ is the density and for a set of discrete data

$$z^* = \frac{\sum_{x_i \in \mathcal{V}_i} \rho(x_i) x_i}{\sum_{x_i \in \mathcal{V}_i} \rho(x_i)}$$

Clearly the generator z_i for $\mathcal{V}_i$ is not equal to z^*.

We call a Centroidal Voronoi Tessellation (CVT) a Voronoi tessellation where $z_i = z^*$, i.e., the generator is the center of mass.



The figure on the left represents a Voronoi tessellation using 10 generators and the figure on the right represents a CVT using 10 generators.

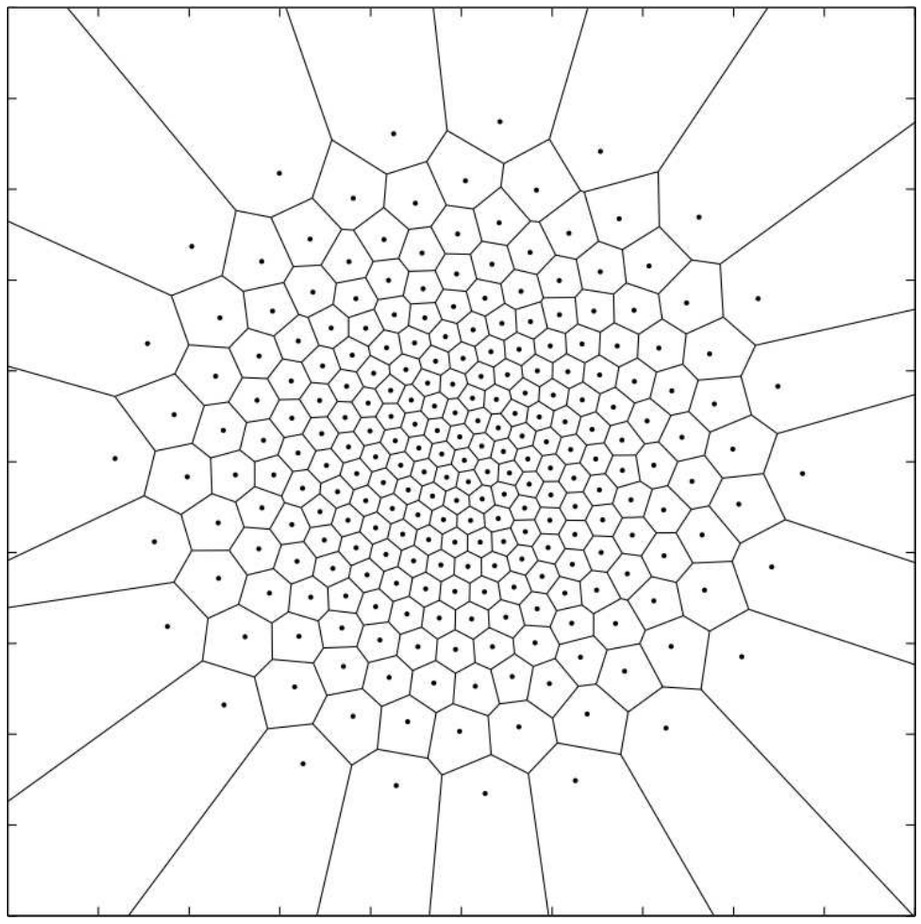
Notice the hexagonal lattice in the interior of the region for the CVT.

The goal of the CVT for the continuous case is to minimize the energy

$$\sum_{i=1}^{K} \int_{y \in \mathcal{V}_i} \rho(y) \|y - z_i\|^2 \, dy$$

It's easy to incorporate weights into the calculation because it is already in the formula for the center of mass.

Here is an example of a CVT using a nonuniform density.



How do we calculate a CVT?

Using Lloyd's algorithm and its variants.

If we have a continuous set of data then we have the following steps.

Step 0 Start with an initial set of K generators $\{z_i\}$, $i = 1, \ldots, K$.

Step 1 Construct the Voronoi tessellation $\{\mathcal{V}_i\}_{i=1}^{K}$.

Step 2 Determine the centers of mass of each Voronoi region $\{\mathcal{V}_i\}_{i=1}^{K}$; move generators to these centers of mass.

Step 3 Check for convergence; if not converged go to Step 1.

In $\mathbb{R}^2$ there is a code called TRIANGLE written by J. Shewchuk to generate a Voronoi tessellation (and the associated Delauney). Other codes exist for finding a Voronoi tessellation on a sphere, etc.

Because the construction of the Voronoi tessellation can be costly, often a probabilistic approach is used.

Probabilistic Lloyd's Algorithm or McQueen's Algorithm

Instead of actually calculating the Voronoi tessellation which is used to calculate the centers of mass we simply sample our domain with lots of points; for each random point we determine which generator it is closest to and keep a tally of the number of points nearest each generator.

Then we determine the center of mass as the average of the points nearest each generator.

One can show that the generators produced by this probabilisitic approach converge to the generators of a centroidal Voronoi tessellation.

This approach is very useful if you are using parallel computers.

When we have discrete data, then we can view CVT as a clustering algorithm. Our algorithm in that case is just Lloyd's algorithm, i.e., the (weighted or unweighted) K-Means algorithm.

CVT Pixel Plot

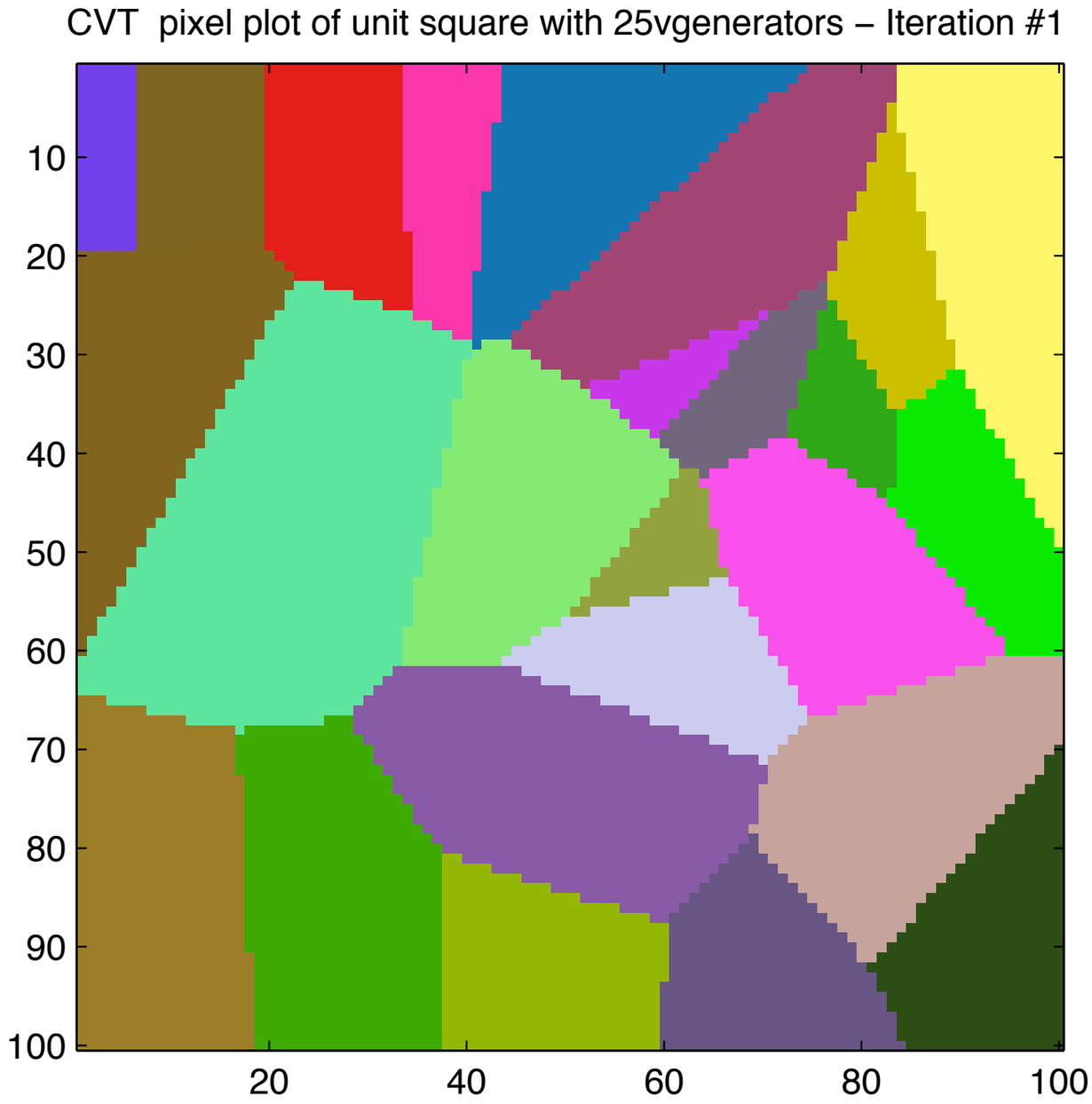
Let's return to the problem of generating our Voronoi pixel plot in the unit square but now let's determine a CVT instead of just a Voronoi tessellation.

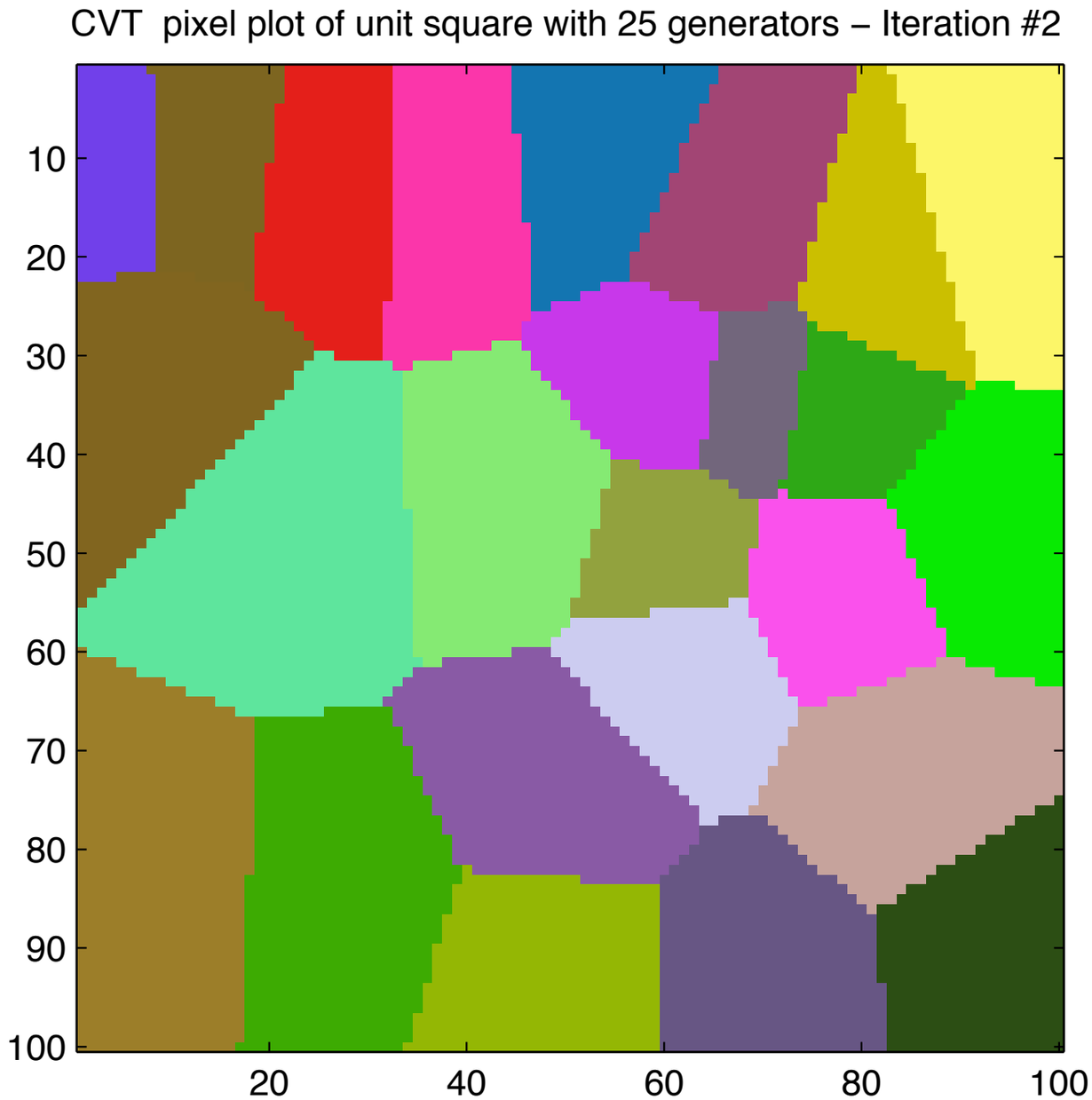
All we have to do is to put an iteration loop around our calculations to perform the Voronoi and then add code to move the generators by averaging the x, y values of each point in that cell.

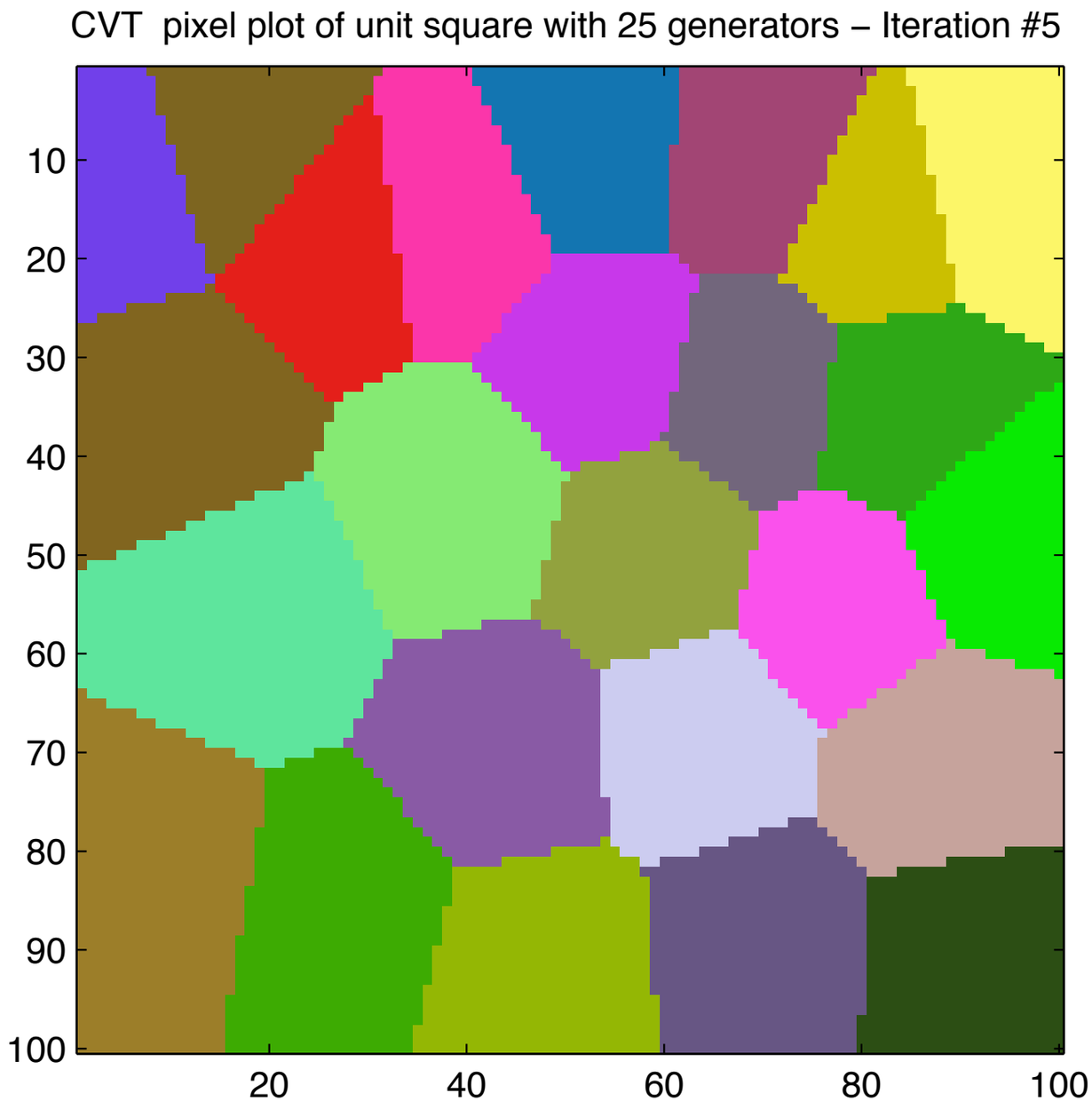
```
for niter=1:maxiters
```

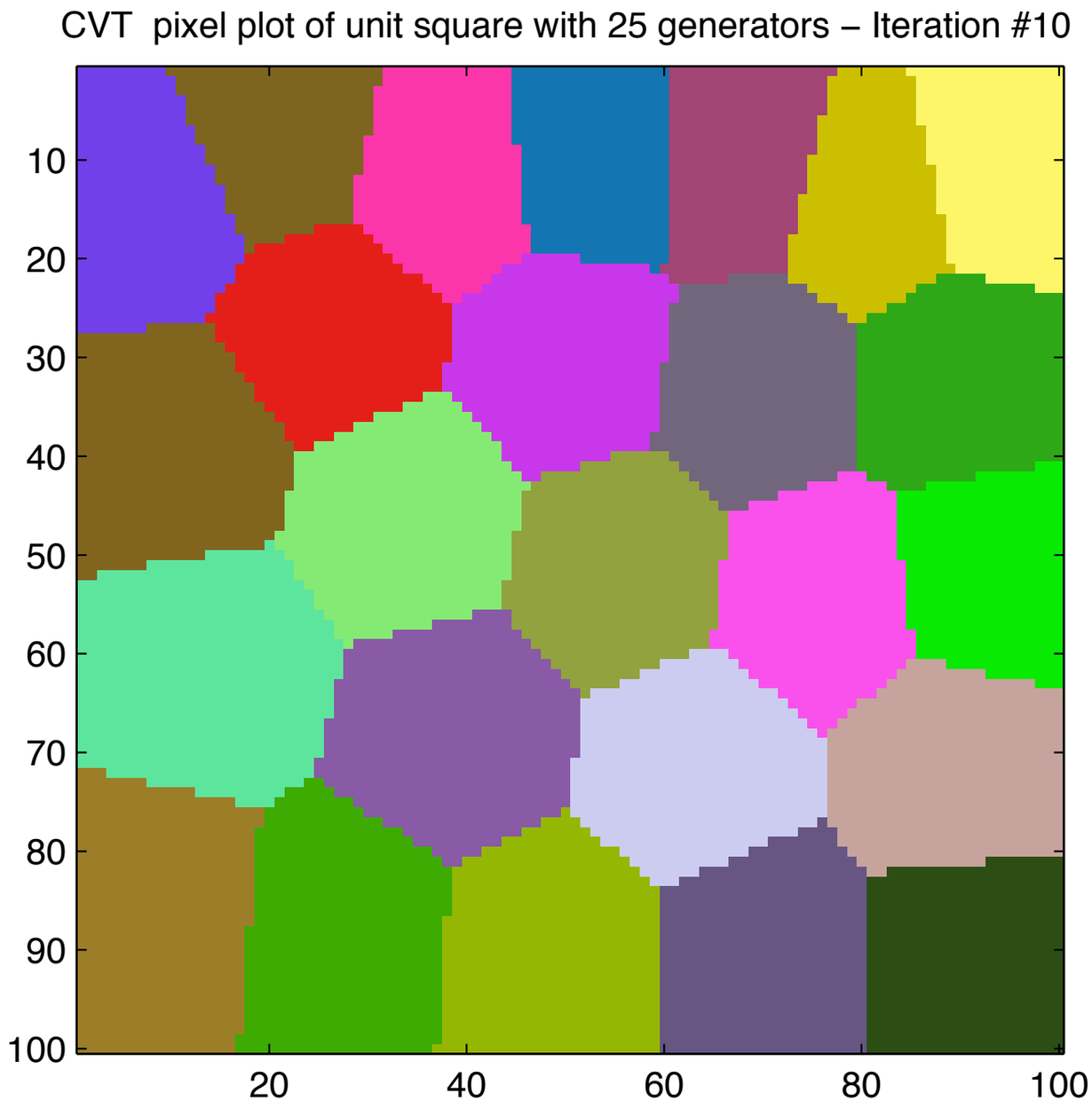
```
 (code as before, except need to zero out znew and count)

znew(1,nearest )=znew(1,nearest )+x;
znew(2,nearest )=znew(2,nearest )+y;
count(nearest) =  count(nearest) + 1;

a(i,j,1:3) = rgb(nearest,1:3);
```

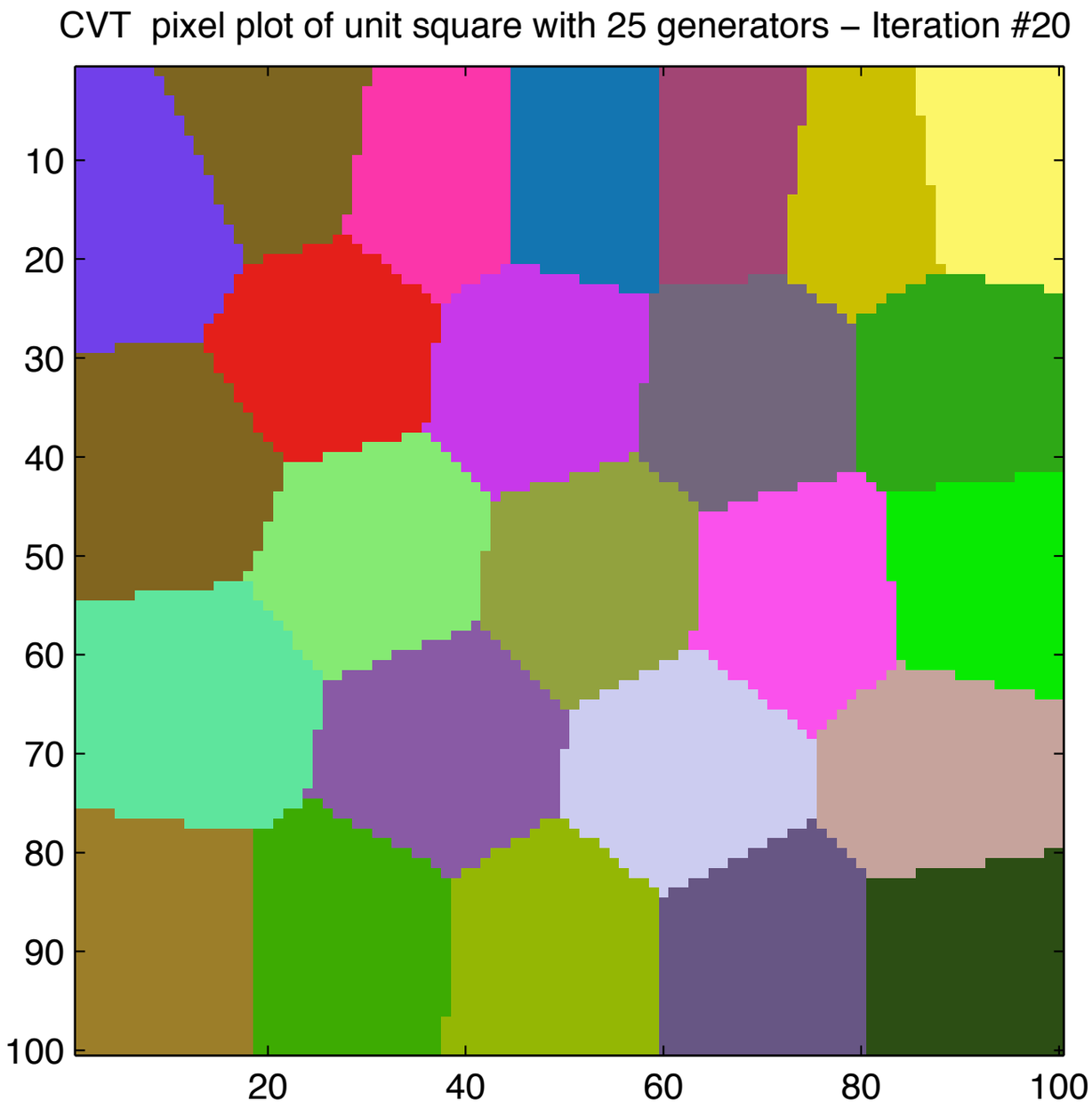
```
    end

  end
%
%  Move the generators
for i=1:nc
    xy(1:2,i) = znew(1:2,i)/ count(i);
end

end  % end iteration loop for CVT
```

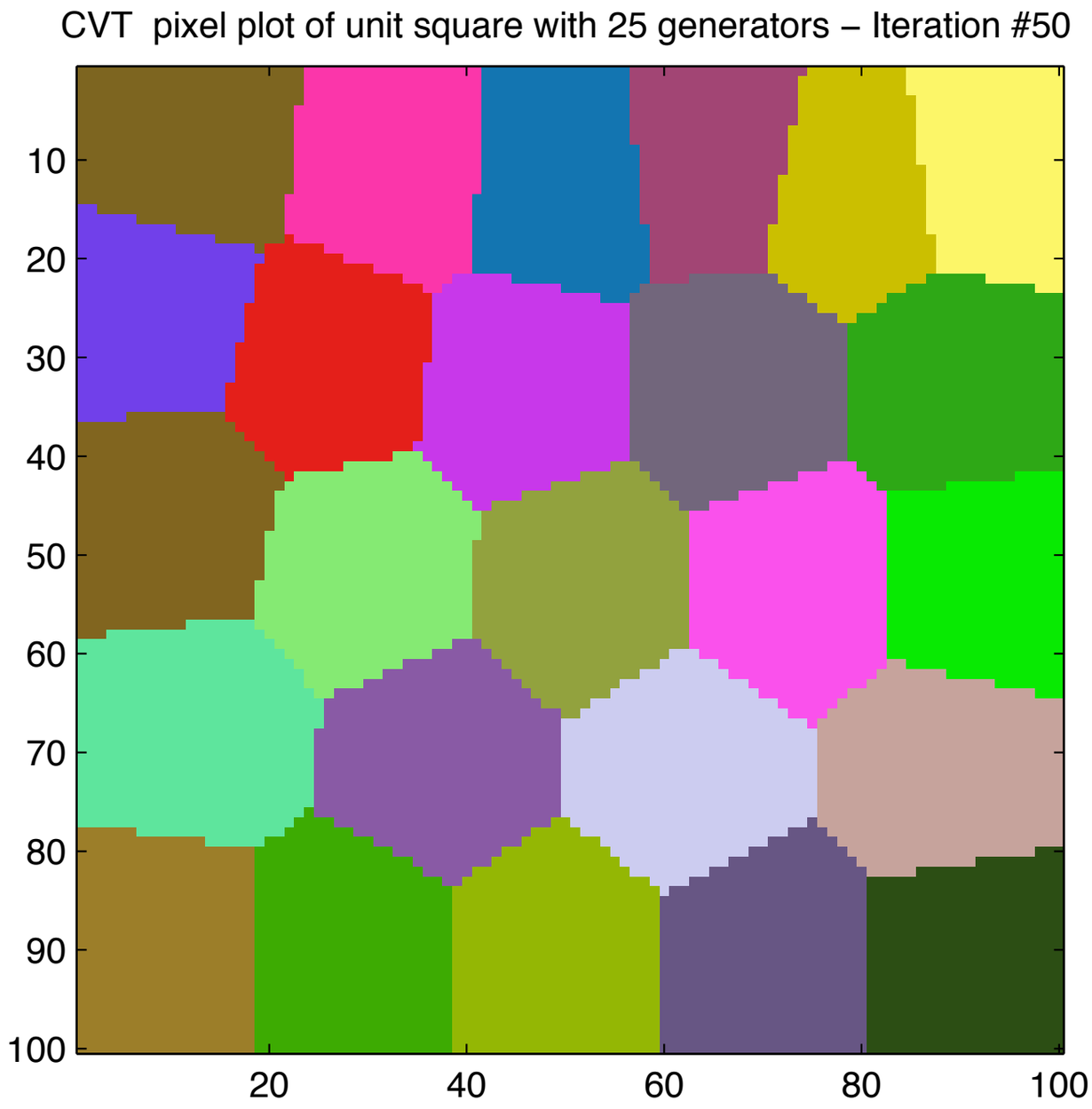

CVT pixel plot of unit square with 25vgenerators – Iteration #1
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100


CVT pixel plot of unit square with 25 generators – Iteration #2
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100


CVT pixel plot of unit square with 25 generators – Iteration #5
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100


CVT pixel plot of unit square with 25 generators – Iteration #10
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100


CVT pixel plot of unit square with 25 generators – Iteration #20
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100


CVT pixel plot of unit square with 25 generators – Iteration #50
10
20
30
40
50
60
70
80
90
100
20
40
60
80
100

Exercise Download the code `voronoi_pixel_plot` and determine a Voronoi plot in the unit square pixel by pixel. Then add code to modify the routine so that it computes a CVT pixel plot.

Data Mining

- Data mining emerged in the 1980's when the amount of data generated and stored became overwhelming.
- Data mining is strongly influenced by other disciplines such as mathematics, statistics, artificial intelligence, data visualization, etc.
- One of the difficulties with it being a new area is that the terminology is not fixed; the same concept may have different names when used in different applications.
- We first see how Data Mining compares with other areas.
- Remember that we are using the working definition:

  "Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data." (W. Frawley).

Data Mining vs Statistics

Statistics can be viewed as a mathematical science and practice of developing knowledge through the use of empirical data expressed in quantitative form. Statistics allows us to discuss randomness and uncertainty via probability theory. For example, statisticians might determine the covariance of two variables to see if these variables vary together and measure the strength of the relationship. But data mining strives to characterize this dependency on a conceptual level and produce a causal explanation and a qualitative description of the data. Although data mining uses ideas from statistics it is definitely a different area.

Data Mining vs Machine Learning

Machine Learning is a subfield of artificial intelligence which emerged in the 1960's with the objective to design and develop algorithms and techniques that implement various types of learning. It has applications in areas as diverse as robot locomotion, medical diagnosis, computer vision, handwriting recognition, etc. The basic idea is to develop a learning system for a concept based on a set of examples provided by the "teacher" and any background knowledge. Main types

are supervised and unsupervised learning (and modifications of these). Machine Learning has influenced data mining but the areas are quite different. Dr. Barbu in the Statistics Department offers a course in Machine Learning.

Data Mining vs Knowledge Discovery from Databases (KDD)

The concept of KDD emerged in the late 1980's and it refers to the broad process of finding knowledge in data. Early on, KDD and Data Mining were used interchangeably but now Data Mining is probably viewed in a broader sense than KDD.

Data Mining vs Predictive Analytics

Wikipedia's definition is " predictive analytics encompasses a variety of techniques from statistics, data mining and game theory that analyze current and historical facts to make predictions about future events." The core of predictive analytics relies on capturing relationships between explanatory variables and the predicted variables from past occurrences and exploiting it to predict future outcomes. One aspect of Data Mining is predictive analytics.

Stages of Data Mining

1. Data gathering, e.g., data warehousing, web crawling
2. Data cleansing - eliminate errors and/or bogus data, e.g., patient fever = 125
3. Feature extraction - obtaining only the interesting attributes of the data, e.g., date acquired is probably not useful for clustering celestial objects
4. Pattern extraction and discovery - this is the stage that is often thought of as data mining
5. Visualization of the data
6. Evaluation of results; not every discovered fact is useful, or even true! Judgement is necessary before following your software's conclusions.

Clearly we can't look at all aspects of Data Mining but we'll just pick a few and get the basic idea.

Dr. Meyer-Baese gives a course in Data Mining if you are interested in learning more about the topic.

- Clustering for feature extraction - we have already talked about this
- Classification - algorithms to assign objects to one of several predefined categories
- Association Rules - algorithms to find interesting associations among large sets of data items.
- Neural Networks
- Support Vector Machine
- Genetic Algorithms

Classification

Examples include:

- classifying email as spam based upon the message header and content
- classifying cells as benign or cancerous based upon results of scan
- classifying galaxies as e.g., spiral, elliptical, etc. based on their shapes
- classifying consumers as potential customers based upon their previous buying

Terminology

Each record is known as an instance and is characterized by the attribute set, $\mathbf{x}$ and a target attribute or class label y

When we use Classification we attempt to find a target function f that maps each attribute set $\mathbf{x}$ to one of the predefined class labels y. The target function is also called the classification model.

Typically we will use a set of data, called the training set, to build our model.

We can use the target function for one of two purposes:

- Descriptive Modeling - Goal is to serve as an explanatory tool to distinguish between objects of different classes.
- Predictive Modeling - Goal is to predict the class label for unknown records.

There are 4 types of attributes:

- nominal - different names; e.g., brown or blue for eye color, SSN, gender

- ordinal - provides ordering; e.g., hot, mild, cold; small, medium, large
- interval - difference in values are meaningful; e.g., dates, temperature
- ratio- differences and ratios are meaningful; e.g., mass, age

Attributes can be discrete or continuous.

Discrete attributes can be categorical such as zip codes, SSN or just numerical. Binary attributes are a special case of discrete and only take on two values such as married or not, homeowner or not, mammal or not, etc. These can be represented as 0 and 1 and are often called Boolean attributes.

Continuous attributes have values which are real numbers; e.g., temperature, weight, salary, etc.

Classification techniques are best suited for data which is binary or nominal. Often when we have continuous data we transform it to ordinal such as small, medium, or large.

General Approach to Solving a Classification Problem

The goal is to use a systematic approach to build a classification model from our training data. The model should fit the training data well and correctly predict the class labels of unseen records not in the training data.

We may use

- decision tree classifiers
- rule-based classifiers
- neural networks
- support vector machine
- Bayes classifiers
- ...

Each technique uses a learning algorithm to identify a model that best fits the

relationship (in some sense) between the attribute set and class label for the input data.

General Steps

1. Provide a training set of records whose class labels are known
2. Apply one of the techniques above to build a classification model using the training set
3. Apply the model to a test set to determine class labels
4. Evaluate the performance of the model based on the number of correct/incorrect predictions of the test set; we can then determine the accuracy as the fraction of correct predictions or the error rate as the fraction of wrong predictions.

Example Suppose we want to classify records as either Class A or Class B. We use our classification model on our test set and get the following results.

Actual Class	Predicted Class	
	Class A	Class B
Class A	43	10
Class B	12	35

In this test set there are 100 records. The table says that 43 records were correctly labeled as Class A and 10 records were incorrectly labeled as Class A. Also 35 Class B records were correctly labeled and 12 were mislabeled as Class A. This means that our accuracy is 78/100 or 78% and our error is 22/100 or 22%.

So now what we need to do is find a way to build a classification model. We will look at decision trees which is probably the easiest approach.

Decision Trees

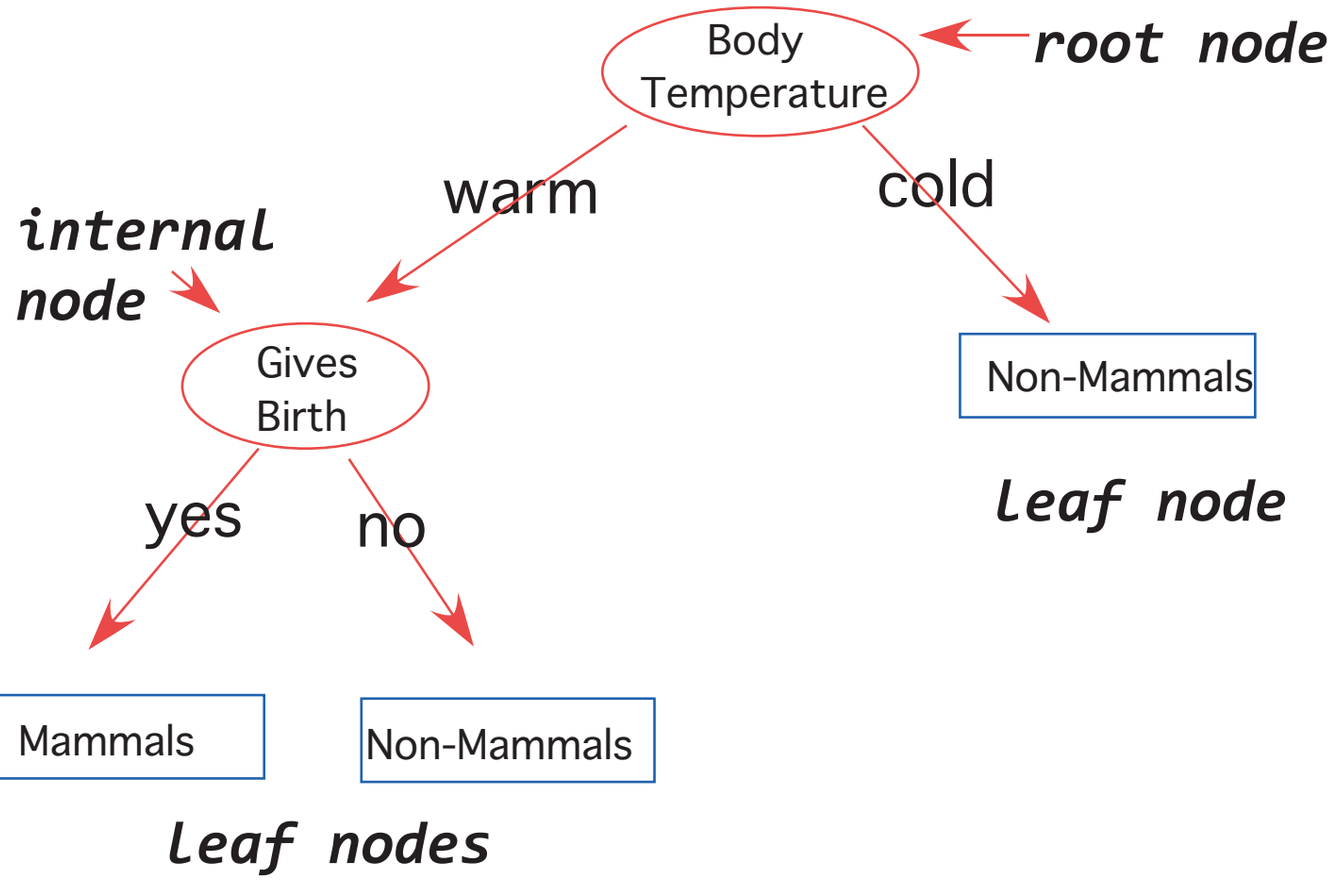
The idea behind decision trees is to pose a series of questions about the characteristics we want. Of course we must carefully choose the questions in order to develop the desired attributes.

Example Suppose we have a list of vertebrates and we want to classify them as mammals or non-mammals. Below is a possible decision tree for classifying a vertebrate. Note the following terminology:

root node - no incoming edges and zero or more outgoing edges

internal node - exactly one incoming edge and two or more outgoing edges

leaf node - exactly one incoming and no outgoing



Suppose we want to use the decision tree to classify a penguin which has the following attributes:

name	body temp	gives birth	class
penguin	warm-blooded	no	?

Applying the decision tree we see that the penguin is classified as a non-mammal because it is warm-blooded but doesn't give birth.

If we think about it we realize that there are exponentially many decision trees that can be constructed from a given set of attributes. So what do we do? Finding the optimal one is probably not an option so we settle for a suboptimal result.

Many decision trees are inductive and use a greedy approach.

A greedy algorithm is one which constructs a solution through a sequence of steps where at each step the choice is made based upon the criteria that

(i) it is the best local choice among all feasible choices available at that step and

(ii) the choice is irrevocable, i.e., it cannot be changed on subsequent steps of the algorithm.

Example Suppose we want to build a decision tree to predict whether a person will default on his/her car loan payments. We collect data from previous bor-

rowers and accumulate the following training set. The attributes we summarize are: (i) homeowner (binary attribute), (ii) marital status (nominal/categorical), (iii) annual income (continuous). Our target class label is binary and is whether that person defaults on the loan payments.

#	home owner	marital status	annual income	defaulted
1	yes	single	125K	no
2	no	married	100K	no
3	no	single	70K	no
4	yes	married	120K	no
5	no	divorced	95K	yes
6	no	married	60K	no
7	yes	divorced	220K	no
8	no	single	85K	yes
9	no	married	75K	no
10	no	single	90K	yes

Hunt's algorithm grows a decision tree in a recursive manner. The records are subsequently divided into smaller subsets until all the records belong to the same

class.

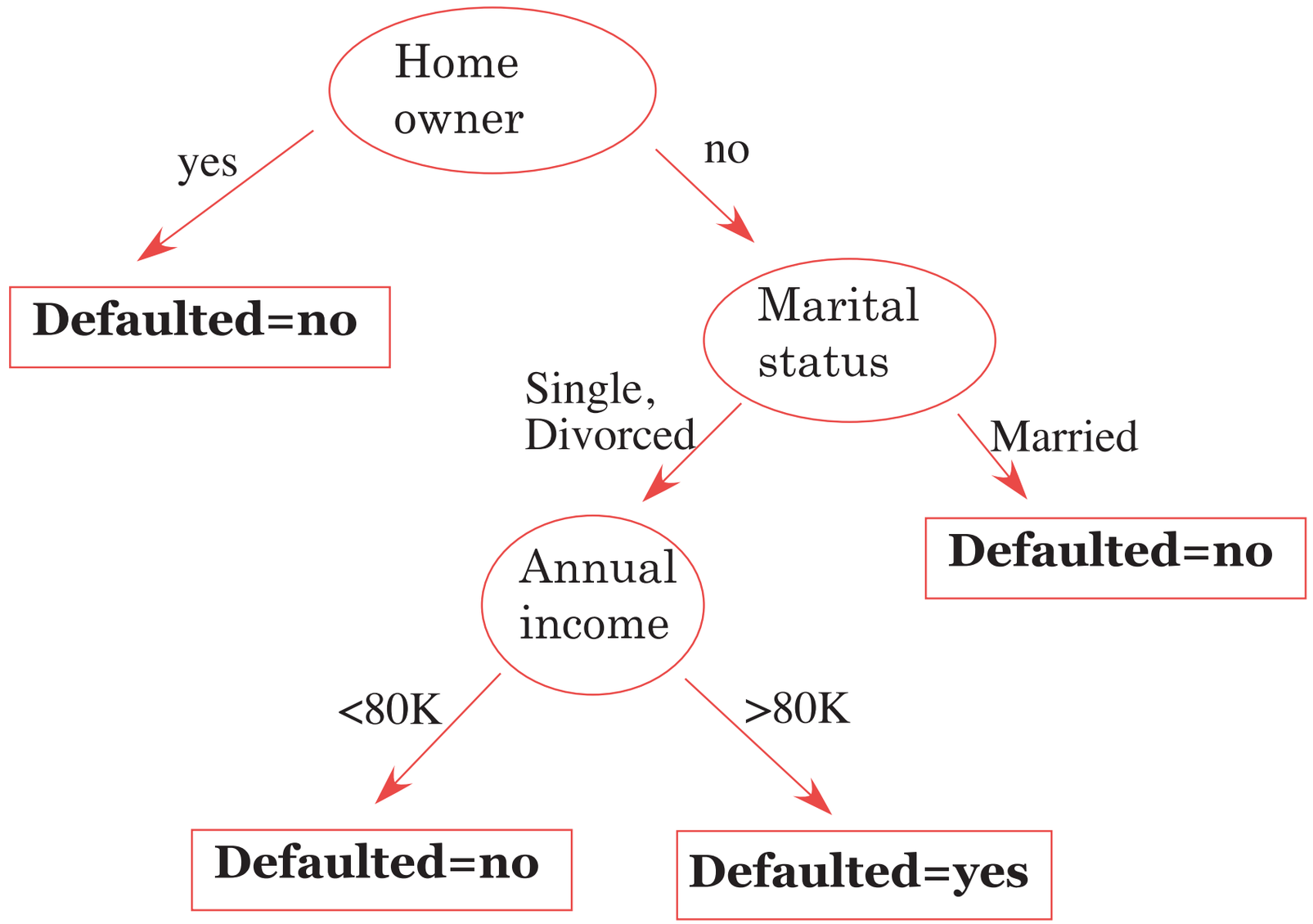
Step 0 - check to make sure all records in training set didn't answer "no" or all answer "yes" to "defaulted". In our training set there were individuals who defaulted and those that didn't.

Step 1 - determine your first criteria for making the "greedy" choice. Here we choose the attributes in order and choose home ownership. We note that all three home owners did not default on their loans so that is a leaf; however some non-home owners defaulted and others didn't so we need to subdivide further.

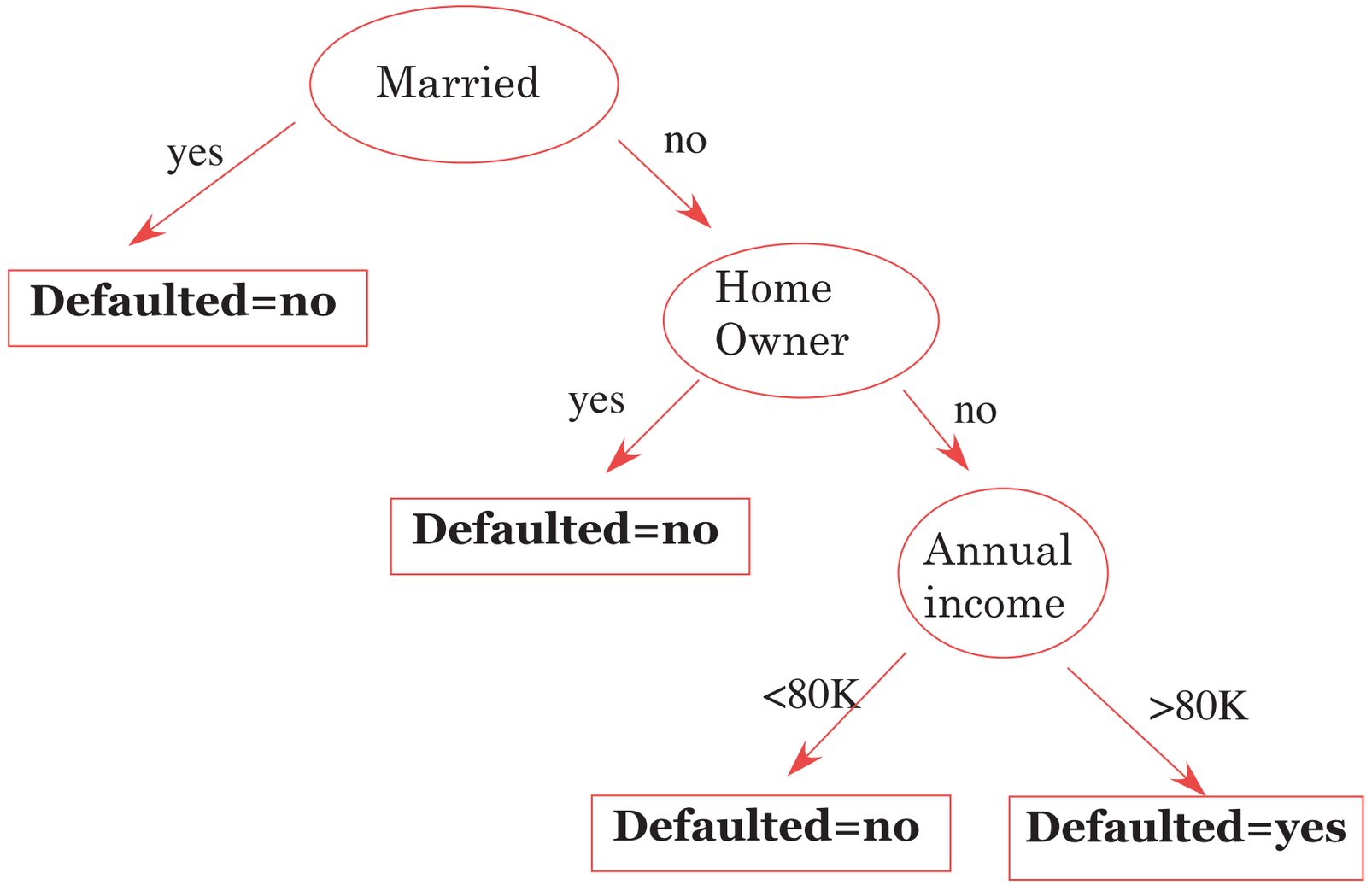
Step 2 - our second criteria is marital status. Here we note that all married borrowers repaid their loans so that is a leaf; however all single and divorced did not repay so we need to subdivide again.

Step 3 - our third criteria is annual income. The group of non-homeowners who are single or divorced is divided by $< 80K$ or $> 80K$. In this case the individuals making more than 80K defaulted and those making less did not.

The resulting decision tree is illustrated below.



Of course if we ask the questions in a different order we get a different decision tree as the following demonstrates.


Married
yes
no
Defaulted=no
Home
Owner
yes
no
Defaulted=no
Annual
income
<80K
>80K
Defaulted=no
Defaulted=yes

Hunt's algorithm for determining a decision tree

We have seen that the approach is recursive and at each step we partition the training records into successively similar subsets.

To describe the method we let $y = \{y_1, y_2, \ldots, y_\ell\}$ be the class labels (in our case we just have default yes and default no). Let $\mathcal{D}_i$ be the ith subset of the training set that is associated with node i (either root, internal or leaf node).

The algorithm is applied recursively as follows:

Check to see if all records in $\mathcal{D}_i$ below to the same class y_i.

- If so, then i is a leaf node (i.e., terminal)
- If not, then choose an attribute test condition to partition the records into smaller subsets. A child node (internal node) is created for each outcome of the test condition and the records in $\mathcal{D}_i$ are distributed according to the outcome of the test condition.

Of course one of these child nodes may be empty if none of the training records have the combination of attributes associated with each node. In this case we just declare it a leaf node with the same class label as the majority class of training records associated with its parent node.

Also suppose we had separated our home owners and the ones who owned homes had identical attributes but different class labels, i.e., some defaulted and some didn't. We couldn't separate these records any further. In this case we declare it a leaf node with the same class label as the majority.

How should we stop the tree growth?

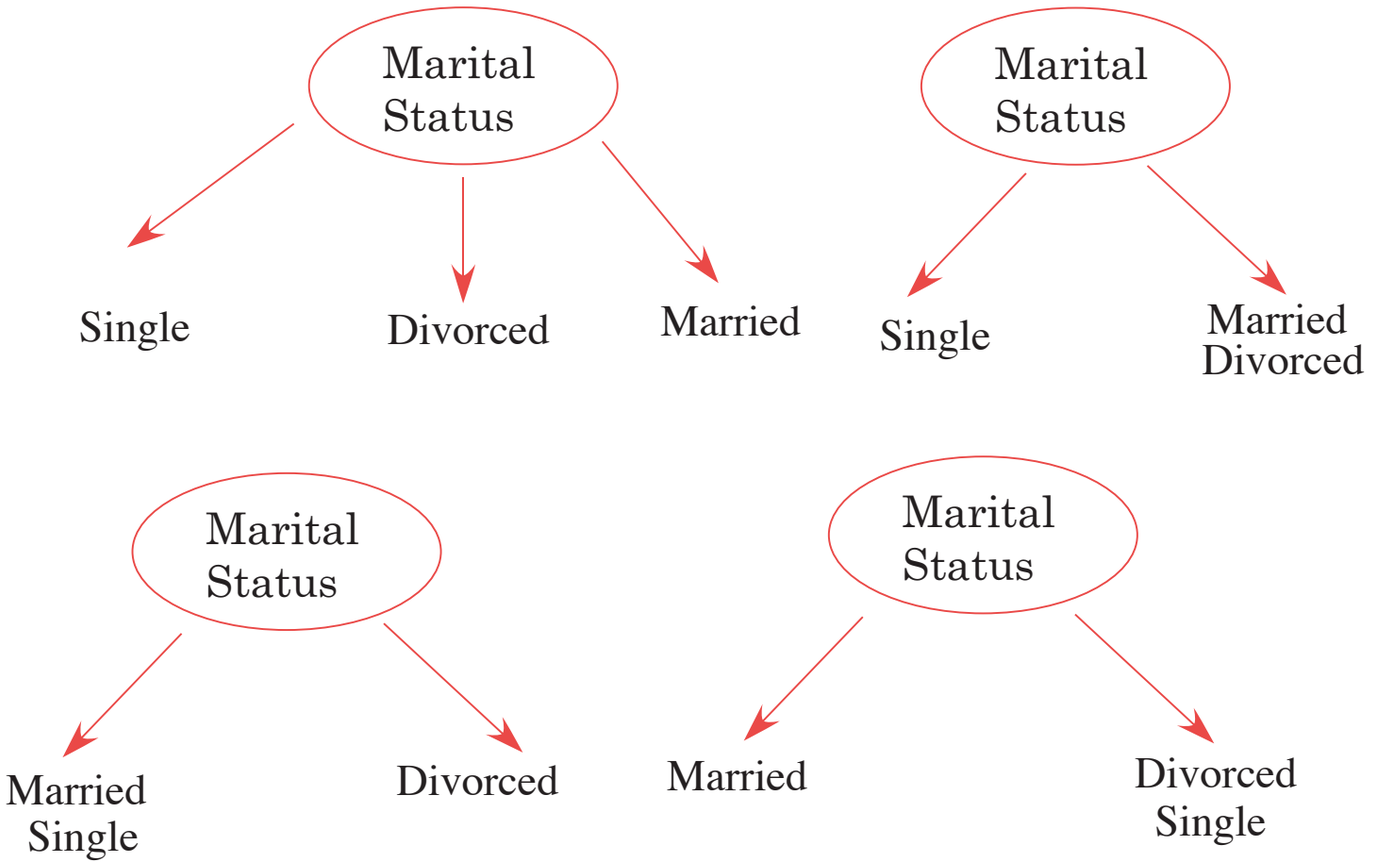
We need a termination criteria for our recursive algorithm. We could stop it when either all records in $\mathcal{D}_j$ have the same class label or are empty or all have identical attributes except for the class label. However, there may be times when it is advantageous to terminate early.

How should we split each training set?

At each recursive step we must select an attribute test condition to divide the records into smaller subsets. So we want to investigate how to choose a test condition to divide $\mathcal{D}_i$. Can we make a more intelligent choice than a random selection? Let's look at the situation for different types of attributes.

Binary attributes are in a sense the easiest because they only generate two potential outcomes; e.g., a home owner query is either yes or no.

Nominal attributes can have many values so splitting them can result in more than two child nodes or we can split it by grouping all but one value in one child node. For example, if we query marital status we can have the following splits.



Ordinal attributes can produce two or more splits; e.g., small, medium, large.

Continuous attributes are usually tested with a comparison, i.e., $\leq$, $>$

So now suppose we are at a step of our algorithm and want to determine which attribute to use as a test. What we would like is a measure for selecting the best

way to divide our records.

Let's look at the easiest case of binary attributes with only two classes (like mammal or non-mammal or default or no default).

Let $p(i|t)$ denote the fraction of records belonging to class i at a given node t; so in the two class problem $p(1) + p(2) = 1$.

When we split $\mathcal{D}_t$ then we would like at least one of the child nodes to be "pure" or homogeneous in the sense that all records in that node are of the same class. So it is reasonable to use a measure of the "impurity" or heterogeneity of the child nodes which we split $\mathcal{D}_t$.

To this end, the following measures are often used for a node t; here k is the number of classes.

Popular Induction Algorithms

1. **Hunt's Algorithm**: this is one of the earliest and it serves as a basis for some of the more complex algorithms.
2. **CART:** classification and regression trees is a non-parametric technique that uses the Gini index to determine which attribute should be split and then the process is continued recursively.
3. **ID3, C4.5**: uses the entropy of an attribute and picks the attribute with the highest reduction in entropy to determine which attribute should the data be split with first and then thro series of recursive functions that calculate the entropy of the node the process is continued until all the left nodes are pure
4. **SLIQ, SPRINT:** are scalable algorithms that have been proposed to deal with the issues the greedy algorithms above present.

Hunt's Algorithm

There are various algorithms that are used to create decision trees. Hunt's Algorithm is one of the earliest and serves as a basis for some of the more complex algorithms. The decision constructed in a recursive fashion until each path ends in a pure subset *(by this we mean each path taken must end with a class chosen).* There are three steps that are done until the fully grown.

1. Examine the record data and find the best attribute for the first node.
2. Split the record data based on this attribute
3. Recurse on each corresponding child node choosing other attributes

Figure 1 is a slide depicting a high level description of Hunt's algorithm.

Issues that arise

There are two issues that arise when dealing with decision tree design algorithms.

1. **How should the training data be split?** We have already seen that by using attribute selection measures we can determine the best node to split for each step.
2. **How should the splitting procedure stop?** At what point do you decide to stop the tree-growing process? This is another crucial area that must not be overlooked. One might that all we need to is to allow the recursive steps play out all the way through to the end but as we will discuss in the next section this might prove counterproductive and might b fruitless.

$$\text{Gini}(t) = 1 - \sum_{i=1}^{k} \big(p(i|t)\big)^2$$

$$\text{Classification error}(t) = 1 - \max_{1 \le i \le k} \big(p(i|t)\big)$$

$$\text{Entropy}(t) = -\sum_{i=1}^{k} \big(p(i|t)\big) \log_2 p(i|t)$$

The first two are related to standard norms. To understand the entropy measure consider the case of two variables like a coin toss. The outcome of a series of coin tosses is a variable which can be characterized by its probability of coming up heads. If the probability is 0.0 (tails every time) or 1.0 (always heads), then there isn't any mix of values at all. The maximum mix will occur when the probability of heads is 0.5 which is the case in a fair coin toss. Let's assume that our measure of mixture varies on a scale from 0.0 ("no mix") to 1.0 ("maximum mix"). This

means that our measurement function would yield a 0.0 at a probability of 0.0 (pure tails), rise to 1.0 at a probability of 0.5 (maximum impurity), and fall back to 0.0 at a probability of 1.0 (pure heads). This is what the Entropy measures does.

Example

Suppose we have 20 records (10 male and 10 female) and our classes are "shop at Overstock.com" or not (say class 1 and class 2) and we have divided the records by gender. For different scenarios of the female "child" node we want to compute the three measurements of error.

(i) all 10 females are of class 1

Because $p(1) = 1.0$ and $p(2) = 0.0$ we have

$$\text{Gini}(t) = 1 - (1^2) = 0$$

$$\text{Classification error}(t) = 1 - 1 = 0.0$$

$$\text{Entropy}(t) = -(1 \log_2(1) + 0) = 0.0$$

and as expected, the "impurity" measures are zero, i.e., the results are homogeneous.

(ii) 5 females are of class 1 and 5 of class 2

Because $p(1) = 0.5$ and $p(2) = 0.5$ we have

$$\text{Gini}(t) = 1 - (.5^2 + .5^2) = 0.5$$

$$\text{Classification error}(t) = 1 - 0.5 = 0.5$$

$$\text{Entropy}(t) = -\big(.5 \log_2(.5) + .5 \log_2(.5)\big) = 1.0$$

These are the maximum values that the measures take on because the class is equally split so it is the least homogeneous.

(ii) 8 females are of class 1 and 2 of class 2

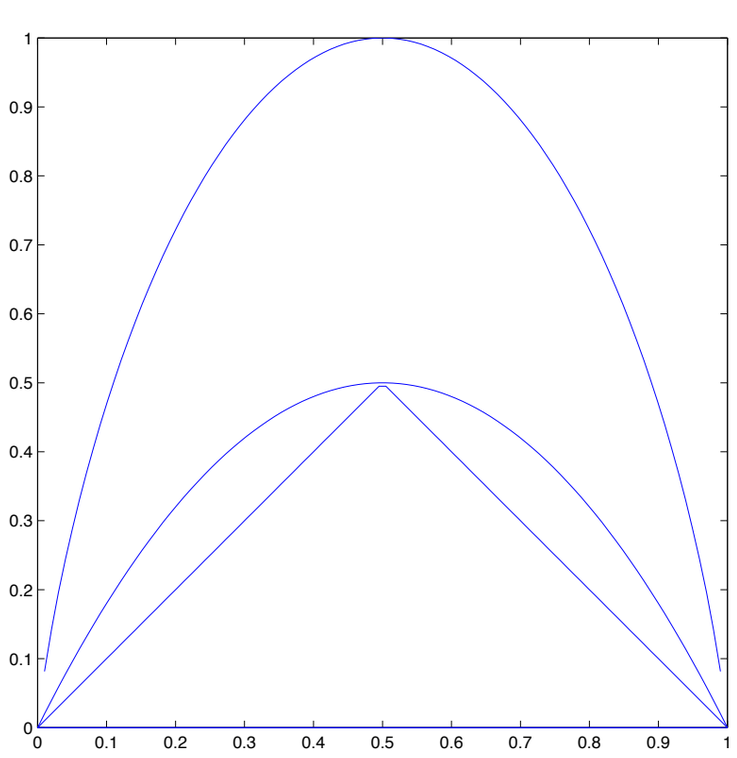
Because $p(1) = 0.8$ and $p(2) = 0.2$ we have

$$\text{Gini}(t) = 1 - (.8^2 + .2^2) = 0.32$$

$$\text{Classification error}(t) = 1 - 0.8 = 0.2$$

$$\text{Entropy}(t) = -\big(.8\log_2(.8) + .2\log_2(.2)\big) = 0.7219$$

If we were to plot these quantities for a range of probabilities we would get that they achieve their maximum when there is a uniform class distribution. This is shown in the figure below.



To determine how a test condition performs, we compare the degree of impurity of the parent node before splitting with the degree of impurity of the child nodes after splitting. The larger their difference the better the test condition.

The gain Δ is a measure that can be used to determine how good a split we are making so our goal will be to choose a test criterion that will maximize the gain. Of course it will depend on the measure we use. Basically all we do is take the difference in the impurity measure of the parent minus a weighted average of the measures of each child node.

Gain: Let I denote the impurity measure (i.e., one of the measurements defined above). Assume that we have split the parent node which consists of N records into k child nodes which consist of N_j records in the jth child node. Then

$$\Delta = I_{\text{parent}} - \sum_{j=1}^{k} \frac{N_j}{N} I_j$$

When the entropy measurement is used it is known as the information gain.

Example Suppose we have a parent node which is equally split so $I_{\text{parent}} = 0.5$ for Gini measure. Now let's say we use two different criteria to split the records and we get two child nodes with the following results.

Criteria A

Node 1 - 4 in class 1 and 3 in class 2

Node 2 - 2 in class 1 and 3 in class 2

Criteria B

Node 1 - 1 in class 1 and 4 in class 2 Node 2 - 5 in class 1 and 2 in class 2

Which criterion is better? Let's use the Gini measure and compare.

We compute the Gini measure for Node 1 to get 0.4898 and for Node 2 we get 0.480 so the gain for using attribute A as a query is the weighted average

$$.5 - \left(\frac{7}{12}(.4898) + \frac{5}{12}(.480)\right) = .5 - 0.4857 = 0.0143$$

For criteria B we compute the Gini measure for Node 1 to get 0.32 and for Node 2 we get 0.4082 so the gain for using attribute B as a query is the weighted average

$$.5 - \left(\frac{5}{12}(.32) + \frac{7}{12}(.4082)\right) = .5 - 0.3715 = 0.128$$

so the gain is higher if we use attribute B to split the parent node. Note that using B results in a smaller weighted average so you get a bigger gain which makes sense because it is a measure of the impurity.

What happens if we use nominal attributes instead of binary to split the records.

If, for example, we have 3 nominal attributes then there are three ways these can be split; two with two child nodes and one with 3 child nodes. For the multiway split (i.e., 3 child nodes) we simply use $k = 3$ in our formula for the gain.

Example Return to our example of the decision tree for deciding whether an individual will default on a loan and decide whether it is better to query (i) home owner or (ii) marital status (assuming 2 child nodes) based on the data given. Use Gini measure. Assume class 1 is "no default."

The parent nodes consists of 10 records 7 of which did not default on the loan so the Gini measure is $1 - .7^2 - .3^2 = 0.420$.

If we use query (i) (home owner) then Node 1 has 3 records all of class 1 and none of class 2 so its Gini measure is 0. Node 2 has 4 records in class 1 and 3 in class 2 so its Gini measure is $1 - (4/7)^2 - (3/7)^2 = 0.4898$. So the gain is 0.42 -(0 +.7(.4898))=0.0771.

If we use query (ii) (marital status) then Node 1 has 4 records all of class 1 so its Gini measure is 0. Node 2 has 3 records in class 1 and 3 in class 2 so its Gini

is 0.5. Thus its gain is 0.42-(0 +.5(.6))=0.12. Thus query (ii) is better by this measure.

Example Here's a canonical example from classification we can investigate. Suppose we have collected the following attributes which we classify as binary, nominal, ordinal, continuous, etc.

ATTRIBUTE	POSSIBLE OUTCOMES
outlook	sunny, overcast, rain (nominal)
temperature	continuous
humidity	continuous
windy	true/false (binary)

Our goal is to predict whether a game (such as tennis) will be played. We use the following training data which consists of 14 records to build our model.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
overcast	83	78	false	Play
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
overcast	64	65	true	Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
rain	75	80	false	Play
sunny	75	70	true	Play
overcast	72	90	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play

Our goal is to determine which decision tree gives the largest information gain using the entropy (randomness) measure. We begin by deciding which attribute

to test first.

1. We start with choosing the outlook as the root node.

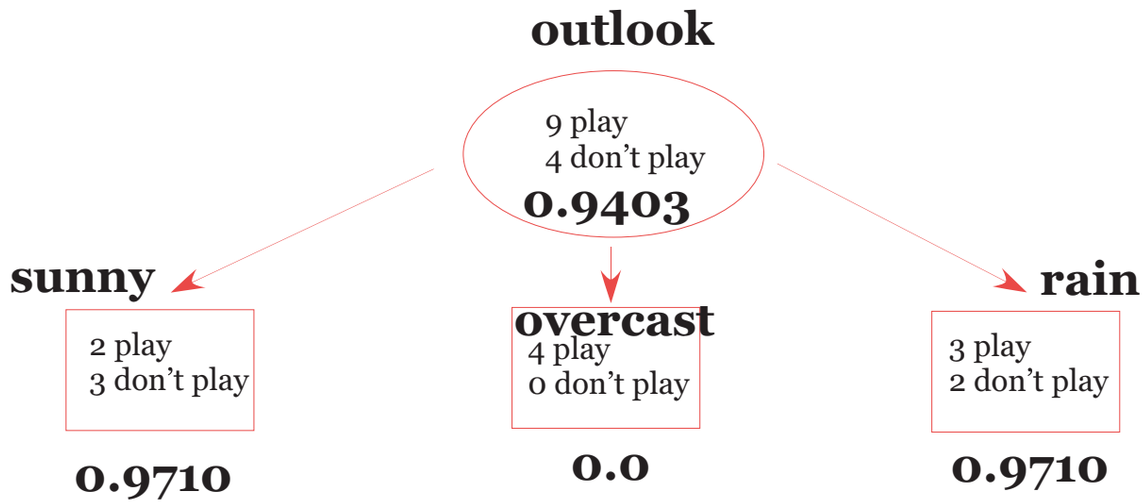
Out of the 14 plays 9 play and 5 don't so we compute the Entropy measure to get

$$-\big((9/14)\log_2(9/14) + (5/14)\log_2(5/14)\big) = 0.9403$$

Similarly the sunny outlook as 5 records, 2 of which play so

$$-\big(.4\log_2 .4 + .6\log_2 .6\big) = 0.9710$$

The overcast child node is homogeneous and so its measure is 0. The rain node has 3 records which play so it has the same measure as the sunny node. This is illustrated below.

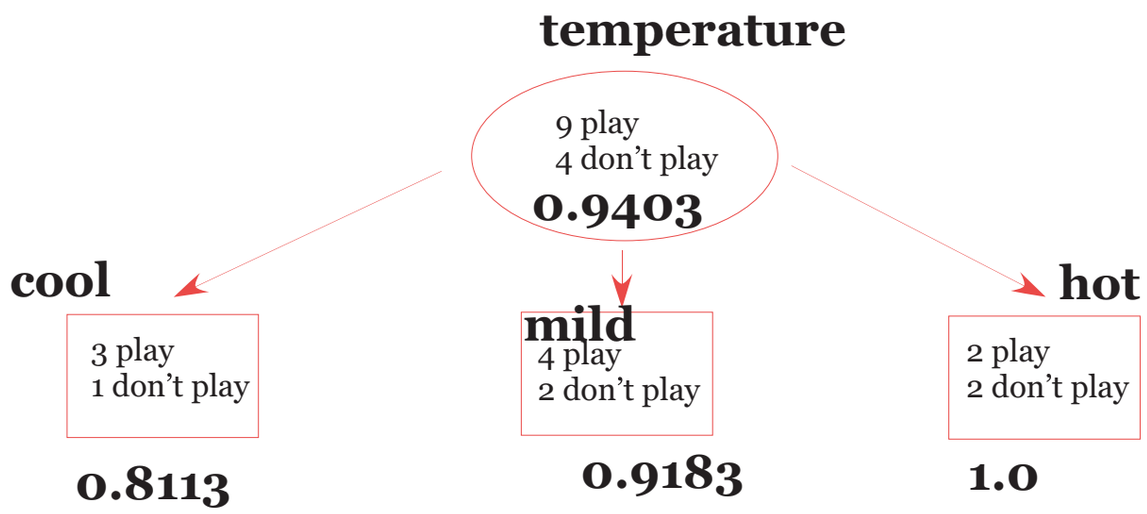


The gain in choosing this is determined by 0.9403 (parent measure) minus a weighted average of the three child nodes, i.e.,

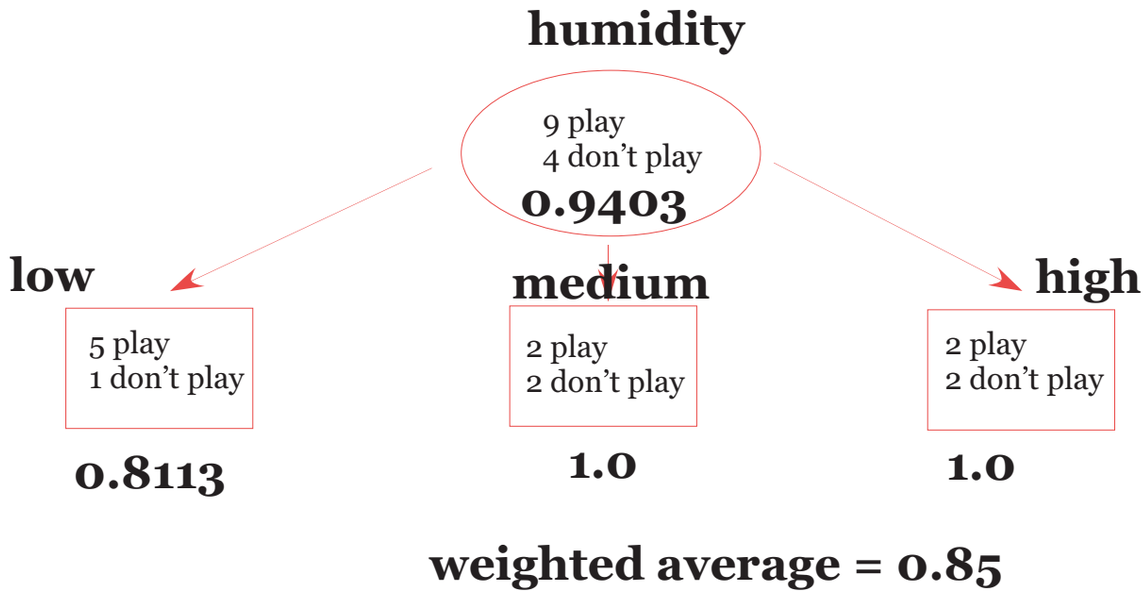
$$\text{information gain} = 0.9403 - \left[\frac{5}{14}.9710 + 0 + \frac{5}{14}.9710\right] = 0.9403 - 0.6929 = 0.2474$$

2. Now we start with the temperature as the root node and compute its gain. For simplicity we break the temperature into cool for temperatures below 70°, mild for temperatures $\geq 70°$ but $< 80°$ and hot for temperatures $\geq 80°$. The Entropy measure for each is shown in the figure and the weighted average for the three child nodes is 0.9111 so the gain is 0.0292 which is less than choosing the outlook as the parent node. We really didn't have to compute the gain, because

the measure for the child nodes was larger than the previous case so it resulted in a smaller gain.



3. Now we start with the humidity as the root node and compute its gain. We divide the humidity as low when it is ≤ 75, medium when it is between 75 and 90 and high for ≥ 90. The measures are given in the figure below and because the weighted average of the measure for the child nodes is 0.85 it is still larger than when we chose outlook as the parent node so the gain is less.



4. Lastly we choose windy as our first attribute to test. This is binary so we only have two child nodes. There are 6 windy records and it was evenly split between play and no play. For the remaining 8 records there were 6 plays and 2 no plays. Clearly the windy node has measure 1.0 and we compute the not windy node measure as 0.8113 so the weighted average is 0.8922 which results in a lower gain.

Consequently we choose outlook as the choice of the first attribute to test. Now we don't have to subdivide the overcast child node because it is homogeneous (pure) but the other two we need to divide. So if we take the sunny node then

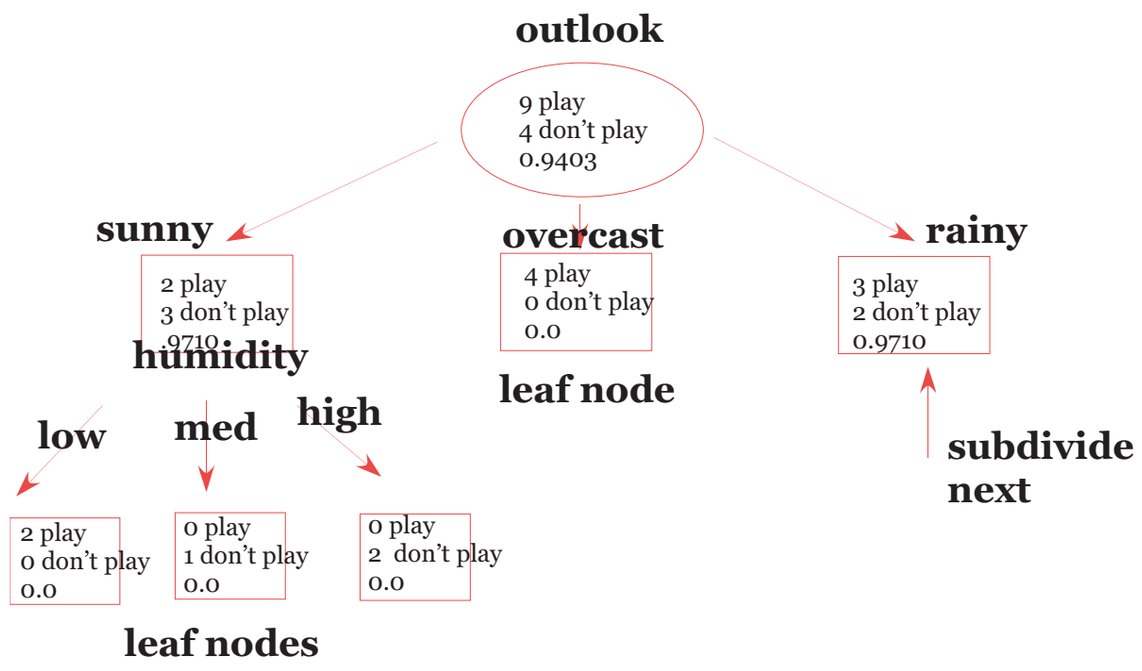
we have to decide whether to test first for temperature, humidity or windy. Then we have to do the same thing for the rainy node.

Here are the 5 sunny records.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
sunny	75	70	true	Play

We tabulate our results for the three remaining attributes below. Clearly the best choice is humidity because it results in all homogeneous child nodes (entropy measure $= 0.0$) so we don't even have to determine the weighted averages to determine the gain.

TEMPERATURE				HUMIDITY				WINDY			
	Play	Don't Play	Measure		Play	Don't Play	Measure		Play	Don't Play	Measure
cool	1	0	0.0	low	2	0	0.0	true	2	1	0.9183
mild	1	1	1.0	med	0	1	0.0	false	0	2	0.0
hot	0	2	0.0	high	0	2	0.0				



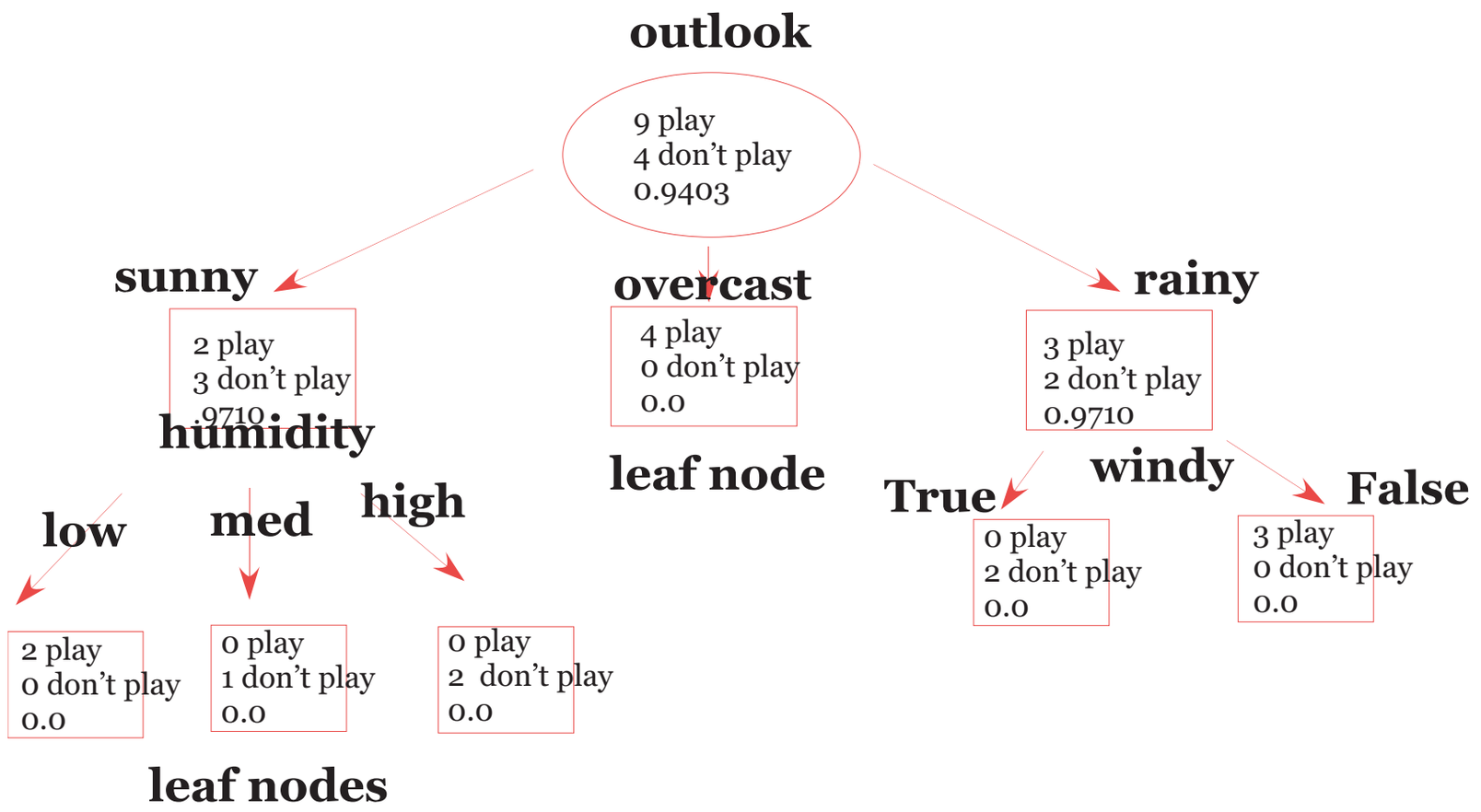
Here are the 5 rainy records which we want to determine how to subdivide. As before we compute the weighted average of our measure for the child nodes and

choose the smallest because it will result in the largest gain.

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	71	80	true	Don't Play

TEMPERATURE				HUMIDITY				WINDY			
	Play	Don't Play	Measure		Play	Don't Play	Measure		Play	Don't Play	Measure
cool	1	1	1.0	low	0	1	0.0	true	0	2	0.0
mild	2	1	0.9183	med	2	1	0.9183	false	3	0	0.0
hot	0	0	0.0	high	0	0	0.0				

Once again we see that windy is the best choice because it results in all the nodes being homogeneous. Our final decision tree using our greedy algorithm with the entropy measure is given below.



Now suppose we have the following two new records to classify using our decision tree. What do you conclude?

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
rain	66	94	true	??
sunny	76	81	false	??

Other Classification Techniques

There are a multitude of classification techniques other than decision trees. We will only briefly look at a few of them due to time constraints.

- Rule-based classifiers
- Nearest neighbor classifiers
- Least Mean squares classifiers
- Bayesian classifiers
- Artificial Neural Networks
- Support Vector Machine
- Ensemble methods

Rule-based Classifier

In Rule-based classification we separate our data records using rules in the form of if – then constructs.

We will use the notation $\wedge$ for "and" and $\vee$ for "or".

To see how this classification technique works, assume that we have a training set of data on vertebrate which has the following attributes and their possible outcomes.

ATTRIBUTE	POSSIBLE OUTCOMES
body temperature	warm- or cold-blooded
skin cover	hair, scales, feathers, quills, fur, none
gives birth	yes/no
aquatic	yes/no/semi
aerial	yes/no
has legs	yes/no
hibernates	yes/no

Our class labels are

mammals, birds, reptiles, fishes, amphibians

From our training set (or from a biology course) we could develop the following rule set $\{r_1, r_2, r_3, r_4, r_5\}$ to classifiy each of the five labels.

$$r_1 : (\text{gives birth} = \text{no}) \wedge (\text{aerial=yes}) \implies \text{bird}$$

$$r_2 : (\text{gives birth} = \text{no}) \wedge (\text{aquatic=yes}) \implies \text{fish}$$

$$r_3 : (\text{gives birth} = \text{yes}) \wedge (\text{body temperature} = \text{warm-blooded}) \implies \text{mammals}$$

$$r_4 : (\text{gives birth} = \text{no}) \wedge (\text{aerial=no}) \implies \text{reptile}$$

$$r_5 : (\text{aquatic} = \text{semi}) \implies \text{amphibian}$$

Now consider two new records which we want to classify

NAME	BODY TEMP	SKIN COVER	GIVES BIRTH	AQUATIC	AERIAL	LEGS	HIBERNATES
grizzly bear	warm	fur	yes	no	yes	yes	yes
turtle	cold	scales	no	semi	no	yes	no
flightless	warm	feathers	no	no	no	yes	no
cormorant							
guppy	cold	scales	ues	yes	no	no	no

Now the grizzly bears does not satisfy the conditions of r_1 or r_2 but r_3 is triggered and it is classified as a mammal.

The turtle triggers r_4 and r_5 but the conclusions of these rules are contradictory so it can't be classified with this rule set.

The flightless cormorant triggers rule 4 so it is incorrectly classified as a reptile.

The last record does not trigger any of the five rules.

We say the rules in a rule set are mutually exclusive if no two rules are triggered by the same record. Thus our rule set above is not mutually exclusive because the record for turtle triggers two rules with contradictory classifications.

We say the rules in a rule set are exhaustive if there is a rule for each combination of the attribute set. Thus our rule set is not exhaustive because it failed to consider the combination of attributes that the guppy has.

Example Can you write a set of mutually exclusive and exhaustive rules which

classify vertebrates as mammals or non-mammals?

Usually one describes the quality of a classification rule by a measure of its accuracy and coverage. If we have a set of records $\mathcal{D}$ and a rule r which classifies data as class y then its coverage is just the fraction of $\mathcal{D}$ that the rule triggers, say $\mathcal{D}_r/\mathcal{D}$. The accuracy or confidence level is just the fraction of records that it classifies correctly, i.e.,

$$\text{accuracy} = \frac{\text{records in } \mathcal{D}_r \text{ which are of class } y}{\mathcal{D}_r}$$

Ordered Rules

Just like when you program if-then constructs the ordering of classification rules can be important. For example, if one rule is expected to have a much higher coverage than the others, we might want to order it first; there are other choices for ranking the rules too. So if we have an ordered set we could avoid the problem we encountered classifying the turtle record. Because we order the rules by some priority we classify a record based on the first rule that it triggers.

Unordered Rules

We could take a different approach and let a record trigger multiple rules. In this case we keep a count of each way we classify the record and then just go with the majority (if there is one). This approach is more computationally intensive because we have to check whether our record satisfies the conditions of each rule whereas in a set of ordered rules we only have to check until the first record is triggered.

Suppose now that we decide to order our rules. What should our strategy be?

If we group all the rules for one class together then we are using class-based ordering.

If we use some quality measure (like coverage) then it is just called rule-based ordering. Remember that when we use ordered rules then when you are at, say

rule r_{10}, you are assuming that the negations of the previous conditions are true so it can often be a bit confusing to interpret a rule.

Here are a few rules in a possible set of class-based ordering for our vertebrate classification. A rule-based ordering set could be any combination of these where we don't group all of our outcomes together.

Class-based ordering

r_1 (aerial=yes) $\wedge$ (skin cover = feathers) $\Longrightarrow$ birds

r_2 (body temp=warm) $\wedge$ (gives birth =no) $\Longrightarrow$ bird

r_3 (body temp=warm) $\wedge$ (gives birth =yes) $\Longrightarrow$ mammal

r_4 (aquatic=semi) $\wedge$ $\Longrightarrow$ amphibian

$\vdots$

Note that this set of rules correctly classifies the flightless cormorant record because it triggers rule r_2 (but not r_1) and it misclassifies the turtle

record because it triggers r_4 and not the previous three rules. However, these rules can't identify the guppy but we haven't added any yet to classify a fish.

Of course the critical question we have not addressed yet is how to build a rule-based classifier. We have to pose our rules in an intelligent manner so that the rules identify key relationships between the attributes of the data and the class label.

Two broad approaches are usually considered. We can look at the training data and try to develop our rules based on it (direct methods) or we can use the results of another classification model such as a decision tree to develop our rules (indirect methods). We consider a direct method here.

Sequential Covering Algorithm

This is a Greedy algorithm which has the strategy that it learns one rule, remove the records it classifies and then repeats the process. One has to specify what criterion to use to order the class labels. So suppose we order our class labels as $\{y_1, y_2, \ldots\}$ and want to develop a rule which classifies y_1 that covers the

training set. All records in the training set that are labeled as class y_1 are considered positive examples and those not labeled as class y_1 are considered negative examples. We seek a rule which covers a majority of the positive examples and none/few of the negative examples.

So basically we just need a routine to learn one rule. Because after we learn this rule we simply remove all records from our training set that are labeled as y_1 and repeat the process again with y_2.

Learn one rule function

Our approach is to "grow" our rule using a greedy strategy. We can either take the approach of starting with a guess for a general rule and then adding conditions to make it more specific or the converse of starting with a specific rule and then pruning it to get a more general rule.

Let's take the general to specific approach for finding a rule to classify mammals. The most general rule is

$$(\quad) \Longrightarrow \text{mammal}$$

which every record satisfies because there is no condition to check. Assume that our training set is given as follows (taken from Tan, et al, Intro to Data Mining).

NAME	BODY TEMP	SKIN COVER	GIVES BIRTH	AQUATIC	AERIAL	LEGS	HIBER-NATES	CLASS
human	warm	hair	yes	no	no	yes	no	mammals
python	cold	scales	no	no	no	no	yes	reptile
salmon	cold	scales	no	yes	no	no	no	fish
whale	warm	hair	yes	yes	no	no	no	mammal
frog	cold	none	no	semi	no	yes	yes	amphibian
k.dragon	cold	scale	no	no	no	yes	no	reptile
bat	warm	hair	yes	no	yes	yes	yes	mammal
robin	warm	feathers	no	no	yes	yes	no	bird
cat	warm	fur	yes	no	no	yes	no	mammals
guppy	cold	scales	yes	yes	no	no	no	fish
alligator	cold	scales	no	semi	no	yes	no	reptile
penguin	warm	feathers	no	semi	no	yes	no	bird
porcupine	warm	quills	yes	no	no	yes	yes	mammal
eel	cold	scales	no	yes	no	no	no	fish
newt	cold	none	no	semi	no	yes	yes	amphibian

So now we want to add a condition to the rule. We look at testing each of the 6 attributes for the 15 records in our training set.

BODY TEMPERATURE = WARM has 7 records satisfying this where 5 are labeled as mammals

BODY TEMPERATURE = COLD has 8 records satisfying this where 0 are labeled as mammals

SKIN COVER = HAIR has 3 records satisfying this with all 3 labeled as mammals

SKIN COVER = QUILLS results in 1 record satisfying it and it is labeled mammal

SKIN COVER = FUR results in 1 record satisfying it and it is labeled mammal

SKIN COVER = SCALES results in 6 record satisfying it and none are labeled mammal

SKIN COVER = FEATHERS results in 2 record satisfying it and none are labeled

mammal

SKIN COVER = NONE results in 2 records satisfying it and none are labeled mammal;

GIVES BIRTH=YES has 6 records satisfying where 5 are labeled as mammals

GIVES BIRTH=NO has 9 records satisfying this but none are labeled as mammals

AQUATIC=YES has 4 records satisfying this with 1 labeled as mammal

AQUATIC=SEMI has 4 records satisfying none are labeled as mammals

AQUATIC=NO has 7 records satisfying this where 4 are labeled as mammals

AERIAL=YES has 2 records satisfying this where 1 is labeled as mammals

AERIAL=NO has 13 records satisfying this where 5 are labeled as mammals

HAS LEGS=YES has 10 records satisfying this where 4 are labeled as mammals

HAS LEGS=NO has 5 records satisfying this where 1 is labeled as mammals

HIBERNATES=YES has 5 records satisfying this where 2 are labeled as mammals

HIBERNATES=NO has 10 records satisfying this where 3 are labeled as mammals

Now let's compute the coverage and accuracy of each which had outcomes labeled as mammal because this is what we are trying to label.

attribute	coverage	accuracy
BODY TEMPERATURE = WARM	7/15	5/7
SKIN COVER = HAIR	3/15	1
SKIN COVER = QUILL	1/15	1
SKIN COVER = FUR	1/15	1
GIVES BIRTH=YES	6/15	5/6
AQUATIC=YES	4/15	1/4
AQUATIC=NO	7/15	4/7
AERIAL=YES	2/15	1/2
AERIAL=NO	13/15	5/13
HAS LEGS=YES	10/15	4/10
HAS LEGS=NO	5/15	1/5
HIBERNATES=YES	5/15	2/5
HIBERNATES=NO	10/15	3/10

Clearly we don't want to go strictly on accuracy because, for example, SKIN COVER = QUILL is completely accurate for our training set but its coverage is only one out of 15 records. If we look at a combination of the coverage and accuracy then the two contenders seem to be (i) BODY TEMPERATURE

= WARM and (ii) GIVES BIRTH=YES. We could choose either based on the criteria we implement. We will discuss criteria shortly.

Once we make our decision we add this to our rule, say we choose BODY TEMPERATURE = WARM. Now we need to test the remaining 6 attributes to make our next choice. Without going through all the possible outcomes let's just assume that GIVES BIRTH=YES results in the best because its coverage and accuracy can be easily determined as 5/7 and 1.0, respectively. To see this note that there are 7 records that satisfy BODY TEMPERATURE = WARM. When we query GIVES BIRTH=YES we get 5/7 coverage with all accurately classified. Thus the rule

$$(\text{BODY TEMPERATURE} = \text{WARM}) \wedge (\text{GIVES BIRTH=YES}) \implies \text{mammal}$$

is our complete rule to classify mammals. We then remove the 5 records in the training set that are labeled mammal and choose our next label y_2 and begin again.

If we take a specific to general approach then we start with the rule

NAME	BODY TEMP	SKIN COVER	GIVES BIRTH	AQUATIC	AERIAL	LEGS	HIBER-NATES	CLASS
human	warm	hair	yes	no	no	yes	no	mammals
python	cold	scales	no	no	no	no	yes	reptile
salmon	cold	scales	no	yes	no	no	no	fish
whale	warm	hair	yes	yes	no	no	no	mammal
frog	cold	none	no	semi	no	yes	yes	amphibian
k.dragon	cold	scale	no	no	no	yes	no	reptile
bat	warm	hair	yes	no	yes	yes	yes	mammal
robin	warm	feathers	no	no	yes	yes	no	bird
cat	warm	fur	yes	no	no	yes	no	mammals
guppy	cold	scales	yes	yes	no	no	no	fish
alligator	cold	scales	no	semi	no	yes	no	reptile
penguin	warm	feathers	no	semi	no	yes	no	bird
porcupine	warm	quills	yes	no	no	yes	yes	mammal
eel	cold	scales	no	yes	no	no	no	fish
newt	cold	none	no	semi	no	yes	yes	amphibian

Name	body temp	skin cover	gives birth	aquatic	aerial	legs	hiber-nates	class
human	warm	hair	yes	no	no	yes	no	mammals
python	cold	scales	no	no	no	no	yes	reptile
salmon	cold	scales	no	yes	no	no	no	fish
whale	warm	hair	yes	yes	no	no	no	mammal
frog	cold	none	no	semi	no	yes	yes	amphibian
k.dragon	cold	scale	no	no	no	yes	no	reptile
bat	warm	hair	yes	no	yes	yes	yes	mammal
robin	warm	feathers	no	no	yes	yes	no	bird
cat	warm	fur	yes	no	no	yes	no	mammals
guppy	cold	scales	yes	yes	no	no	no	fish
alligator	cold	scales	no	semi	no	yes	no	reptile
penguin	warm	feathers	no	semi	no	yes	no	bird
porcupine	warm	quills	yes	no	no	yes	yes	mammal
eel	cold	scales	no	yes	no	no	no	fish
newt	cold	none	no	semi	no	yes	yes	amphibian

$($ BODY TEMPERATURE $=$ WARM $) \wedge$ (SKIN COVER $=$ HAIR $) \wedge ($ GIVES BIRTH$=$YES $) \wedge ($ AQUATIC$=$NO $) \wedge ($ AERIAL $=$ NO $) \wedge ($ HAS LEGS$=$YES $) \wedge ($ HIBERNATES $= 0$ $) \implies$ mammal

There is only one record (human) in the training set that has all of these attributes so clearly we need to remove some. The procedure is analogous to before.

What criteria should we use to add a rule?

We have seen that accuracy is not enough, because, e.g., we had complete accuracy for the rule SKIN COVER $=$ QUILL but there was only 1 positive example of this in our training set. Coverage alone is not enough because, e.g., HIBERNATES$=$NO had 10 positive examples in the training set but was only 30% accurate. Here are a couple of commonly used evaluation metrics.

Let n denote the number of records which satisfy the condition of the rule, n_+ denote the number of positive records (i.e., the ones for which the outcome is

true) and let k denote the number of classes. Then

$$\text{Laplace} = \frac{n_+ + 1}{n + k}$$

Let's look at this measure for two of our examples above.; here $k = 5$ (mammals, fish, reptiles, amphibians, bird).

SKIN COVER = QUILL Laplace $= \frac{2}{20} = .1$

HIBERNATES=NO Laplace $= \frac{4}{15} = .266$

Now let's compare two attributes and compare. Recall that BODY TEMPERATURE = WARM had a coverage of 7/15 and an accuracy of 5/7

SKIN COVER = QUILL has a coverage of 1/15 and an accuracy of 1.0

The second one has a better accuracy but lower coverage. Let's compute the Laplace measure for each.

One well-known formula arising from his system is the rule of succession, given as principle seven. Suppose that some trial has only two possible outcomes, labeled "success" and "failure". Under the assumption that little or nothing is known *a priori* about the relative plausibilities of the outcomes, Laplace derived a formula for the probability that the next trial will be a success.

$$\Pr(\text{next outcome is success}) = \frac{s+1}{n+2}$$

where s is the number of previously observed successes and n is the total number of observed trials. It is still used as an estimator for the probability of an event if we know the event space, but have only a small number of samples.

The rule of succession has been subject to much criticism, partly due to the example which Laplace chose to illustrate it. He calculated that the probability that the sun will rise tomorrow, given that it has never failed to in the past, was

$$\Pr(\text{sun will rise tomorrow}) = \frac{d+1}{d+2}$$

where d is the number of times the sun has risen in the past. This result has been derided as absurd, and some authors have concluded that all applications of the Rule of Succession are absurd by extension. However, Laplace was fully aware of the absurdity of the result; immediately following the example, he wrote, "But this number [i.e., the probability that the sun will rise tomorrow] is far greater for him who, seeing in the totality of phenomena the principle regulating the days and seasons, realizes that nothing at the present moment can arrest the course of it."[38]

Pierre-Simon Laplace

Pierre-Simon Laplace (1749–1827). Posthumous portrait by Madame Feytaud, 1842.

Born	23 March 1749 Beaumont-en-Auge, Normandy, France
Died	5 March 1827 (aged 77) Paris, France
Nationality	French
Fields	Astronomer and Mathematician
Institutions	École Militaire (1769–1776)
Alma mater	University of Caen
Academic advisors	Jean d'Alembert Christophe Gadbled Pierre Le Canu
Doctoral students	Siméon Denis Poisson
Known for	Work in Celestial Mechanics Laplace's equation Laplacian Laplace transform Laplace distribution Laplace's demon

$\text{BODY TEMPERATURE} = \text{WARM} \quad \text{Laplace} = \frac{5+1}{7+5} = .5$

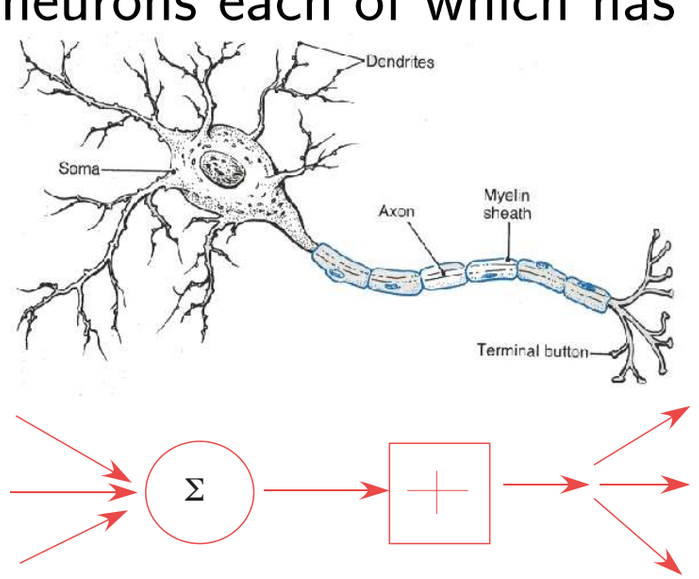
$\text{SKIN COVER} = \text{QUILL} \quad \text{Laplace} = \frac{1+1}{1+5} = .3333$

If we compare this with the accuracy the second test has a much smaller value of its Laplace measure than its accuracy so this says the accuracy of 1.0 was spurious because the test didn't has enough positive records.

Clearly one can think of many other metrics or combinations thereof to use.

Neural Networks (NN)

This idea was inspired by attempts to simulate a biological neural system where neurons are linked together via axons which are used to transmit nerve impulses from one neuron to another. Neurons are connected to axons of other neurons via dendrites. The connection between the dendrite and axon is called a synapse and we learn by changing the strength of this connection. It is estimated that the human brain has 10^{10} neurons each of which has thousands of connectors.



Neural Networks have seen an explosion of interest over the last few years, and are being applied across a range of problems as diverse as finance, medicine, engineering, geology and physics. Neural Networks are used where problems involve prediction, classification or control.

For many years linear modeling (like linear regression) was the commonly used technique in most modeling because linear models are simple to use and have well-known optimization strategies. Of course if a linear approximation is not valid (which is frequently the case) the models are not representative of the data.

Neural networks are very sophisticated modeling techniques capable of modeling complex functions. In particular, neural networks can be nonlinear. Neural networks also keep in check the "curse of dimensionality".

Neural networks learn by example. The neural network user gathers representative data, and then invokes training algorithms to automatically learn the structure of the data. Although the user does need to have some heuristic knowledge of how to select and prepare data, how to select an appropriate neural network, and how to interpret the results, the level of user knowledge needed to successfully

apply neural networks is much lower than would be the case using some other methods.

Some Applications of Neural Networks

- Detection of medical phenomena A variety of health-related indices (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a complex (e.g., nonlinear and interactive) combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed.
- Stock market prediction Fluctuations of stock prices and stock indices are complex and multidimensional, but in some circumstances at least partially-deterministic phenomenon. Neural networks are being used by many technical analysts to make predictions about stock prices based upon a large number of factors such as past performance of other stocks and various economic indicators.

- Credit assignment A variety of pieces of information are usually known about an applicant for a loan. For instance, the applicant's age, education, occupation, and many other facts may be available. After training a neural network on historical data, neural network analysis can identify the most relevant characteristics and use those to classify applicants as good or bad credit risks.

- Monitoring the condition of machinery Neural networks can be instrumental in cutting costs by bringing additional expertise to scheduling the preventive maintenance of machines. A neural network can be trained to distinguish between the sounds a machine makes when it is running normally ("false alarms") versus when it is on the verge of a problem. After this training period, the expertise of the network can be used to warn a technician of an upcoming breakdown, before it occurs and causes costly unforeseen down-time.

Now to describe neural nets we want to capture the essence of biological neural systems so we define an artificial neuron as follows:

- It receives a number of inputs (either from original data, or from the output

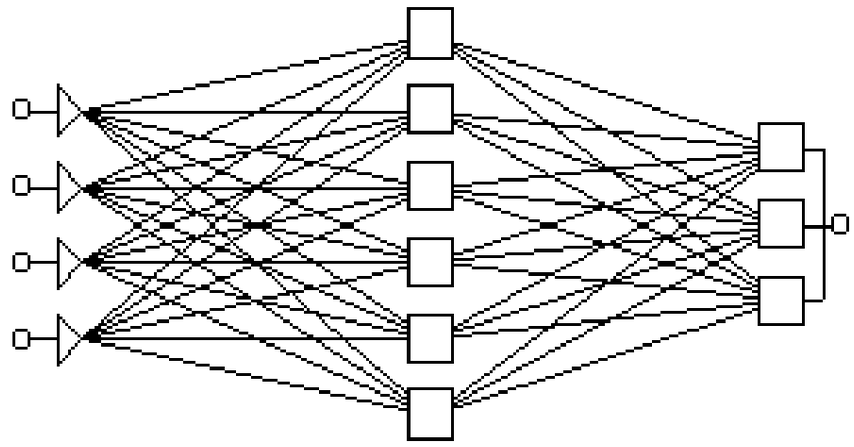
of other neurons in the network).

- Each input comes via a connection that has a strength (i.e., a weight); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to determine the activation of the neuron.
- The activation signal is passed through an activation function (also known as a transfer function) to produce the output of the neuron.

Now we have to decide how to connect the neurons.

- If a network is to be of any use, there must be inputs (which carry the values of variables of interest) and outputs (which form predictions). Inputs and outputs correspond to sensory and motor nerves such as those coming from the eyes and leading to the hands.
- A simple network has a feedforward structure: signals flow from inputs, forward through any hidden units, eventually reaching the output units. Such a structure has stable behavior.
- A typical feedforward network has neurons arranged in a distinct layered

topology. The input layer is not really neural at all: these units simply serve to introduce the values of the input variables. The hidden and output layer neurons are each connected to all of the units in the preceding layer.



Example of a Simple Multilayer Neural Network

To see how we can use Neural Networks we consider an example from pattern recognition. Before we do this we will look at a simple case of firing a neuron. We won't be going into the details but you can get an idea how it could work. The reference for examples is Neural Networks by Christos Stergiou and Dimitrios Siganos.

The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained.

Take a collection of training patterns for a node, some of which cause it to fire (the "1" taught set of patterns) and others which prevent it from doing so (the "0" taught set). Then the patterns not in the collection cause the node to fire if, on comparison , they have more input elements in common with the "nearest"

pattern in the "1"-taught set than with the "nearest" pattern in the "0"-taught set. If there is a tie, then the pattern remains in the undefined state.

As an example, consider a 3-input neuron which is taught to output 1 when the input (X1,X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001. Before we apply the firing rule, we consider the following table.

```
X1:   0 0   0    0    1   1    1   1
X2:   0 0   1    1    0   0    1   1
X3:   0 1   0    1    0   1    0   1
OUT: 0 0 0/1 0/1 0/1 1   0/1 1
```

The four training inputs are listed in this table (e.g., if the input is 000 the output is 0 or if the input is 101 then the output is 1) but other combinations are listed with output either 0 or 1 – to be determined.

As an example of the way the firing rule is applied, take the input 010. It differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from

111 in 2 elements. Therefore, the 'nearest' pattern is 000 which belongs in the 0-taught set. Thus the firing rule requires that the neuron should not fire when the input is 001. On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1).

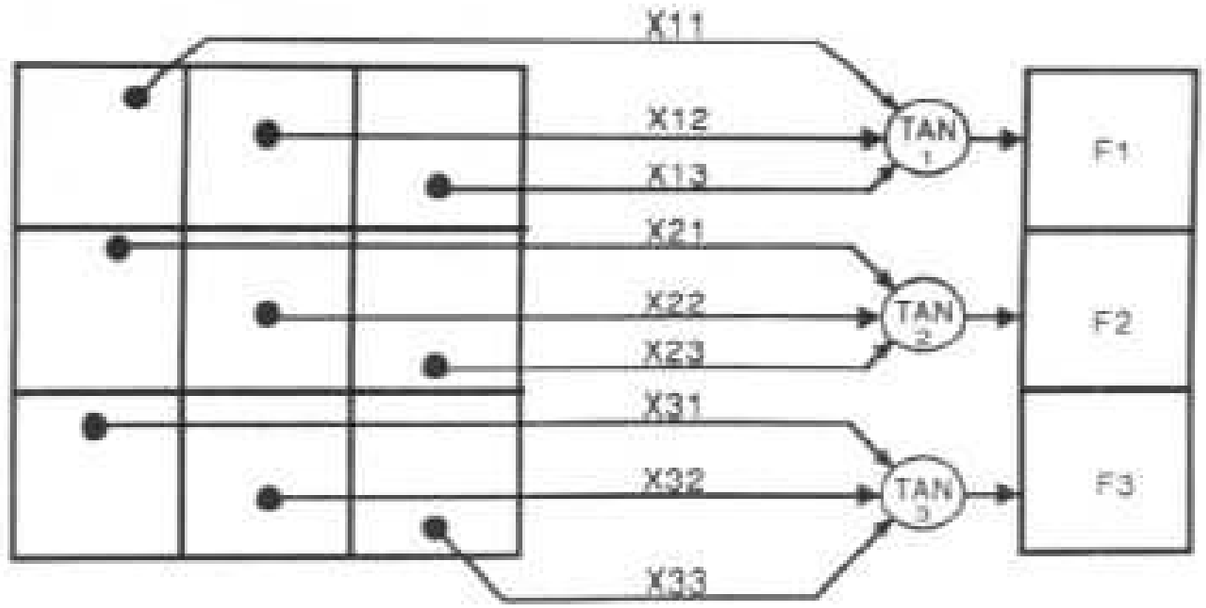
By applying the firing in every column the following truth table is obtained:

```
X1:   0 0 0 0 1 1 1 1
X2:   0 0 1 1 0 0 1 1
X3:   0 1 0 1 0 1 0 1
OUT: 0 0 0 0/1 0/1 1 1 1
```

The difference between the two truth tables is called the generalization of the neuron. Therefore the firing rule gives the neuron a sense of similarity and enables it to respond "sensibly" to patterns not seen during training.

An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward neural network that has been trained accordingly. During training, the network is trained to associate outputs with input patterns. When the network is used, it identifies the input

pattern and tries to output the associated output pattern. The power of neural networks comes to life when a pattern that has no output associated with it, is given as an input. In this case, the network gives the output that corresponds to a taught input pattern that is least different from the given pattern.



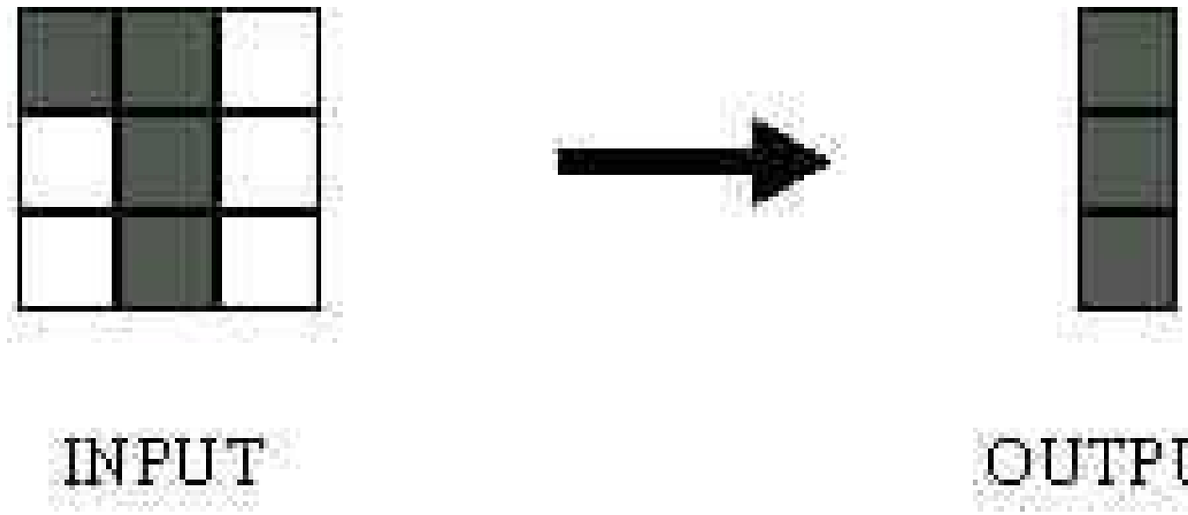
Assume that the network shown above is trained to recognise the patterns “T” and “H”. The associated patterns are all black and all white respectively as shown below.

If we represent black squares with 0 and white squares with 1 then the truth tables for the 3 neurons after generalization can be written as

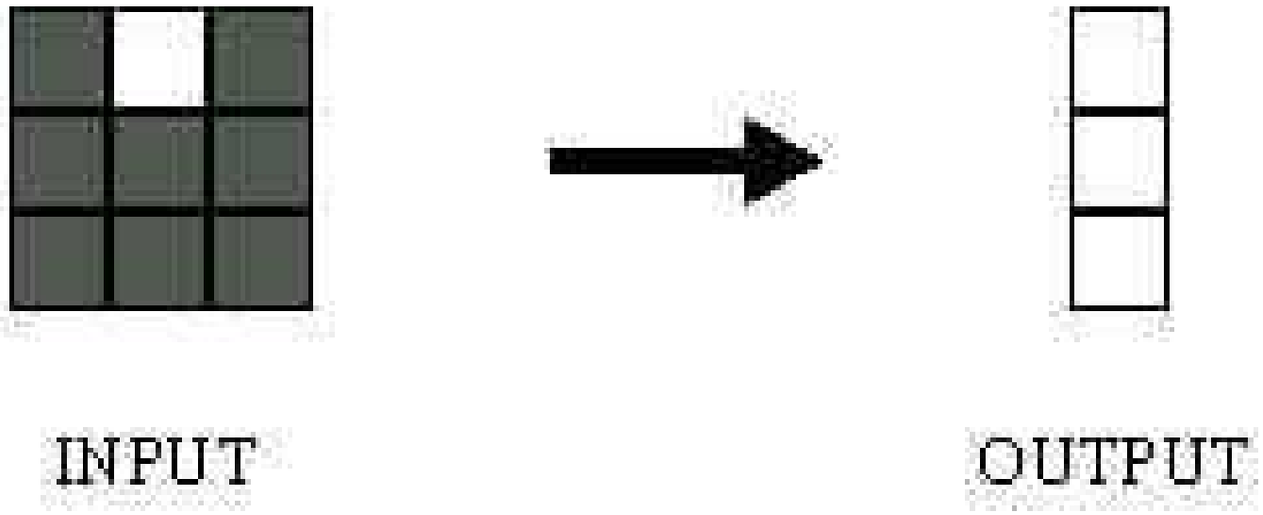
```
X11: 0 0 0 0 1 1 1 1
X12: 0 0 1 1 0 0 1 1
X13: 0 1 0 1 0 1 0 1
OUT: 0 0 1 1 0 0 1 1
 Top neuron

X21: 0  0  0  0   1  1  1  1
X22: 0  0  1  1   0  0  1  1
X23: 0  1  0  1   0  1  0  1
OUT: 1 0/1 1 0/1 0/1 0 0/1 0
 Middle neuron
```

```
X21: 0 0 0 0 1 1 1 1
X22: 0 0 1 1 0 0 1 1
X23: 0 1 0 1 0 1 0 1
OUT: 1 0 1 1 0 0 1 0
 Bottom neuron
```

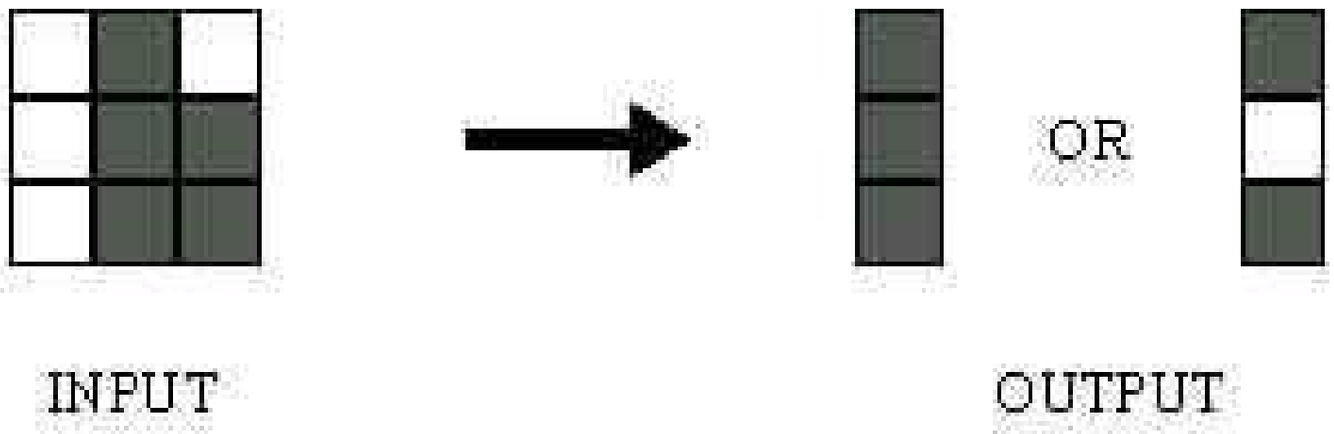
From the tables we can get the following associations:



In this case, it is obvious that the output should be all blacks since the input pattern is almost the same as the "T" pattern.



Here it is obvious that the output should be all whites since the input pattern is almost the same as the "H" pattern.



Here, the top row is 2 errors away from the a T and 3 from an H. So the top

output is black. The middle row is 1 error away from both T and H so the output is random. The bottom row is 1 error away from T and 2 away from H. Therefore the output is black. The total output of the network is still in favour of the T shape.