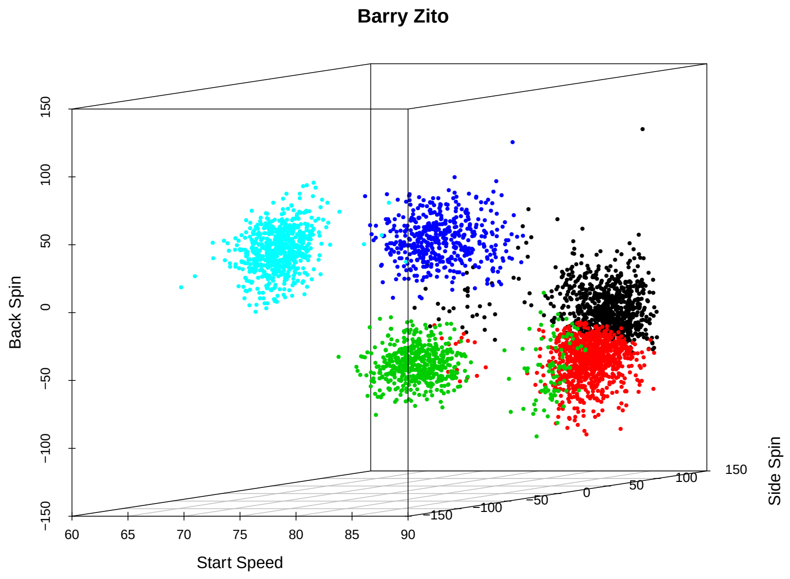
Clustering

Advanced Methods for Data Analysis (36-402/36-608)

Spring 2014

1 Introduction to clustering

- *Clustering* is the task of dividing up data points into groups or clusters, so that points in any one group are more "similar" to each other than to points outside the group
- Why cluster? Two main uses:
  - *Summary*: deriving a reduced representation of the full data set. E.g., vector quantization (we'll see this shortly)
  - *Discovery*: looking for new insights into the structure of the data. E.g., finding groups of students that commit similar mistakes, groups of 80s songs that sound alike, or groups of patients that have similar gene expressions
- There are other uses too, e.g., checking up on someone else's work/decisions, by investigating the validity of pre-existing group assignments; or helping with prediction, i.e., in regression or classification. For brevity, we won't study clustering in these contexts
- A neat example: clustering of baseball pitches



Inferred meaning of clusters: black – fastball, red – sinker, green – changeup, blue – slider, light blue – curveball. (This example is due to Mike Pane, a former CMU statistics undergrad, from his undergraduate honors thesis)

- A note: don't confuse clustering and classification! In classification, we have data points for which the groups are *known*, and we try to learn what differentiates these groups (i.e., a classification function) to properly classify future data. In clustering, we look at data points for which groups are *unknown and undefined*, and try to learn the groups themselves, as well as what differentiates them. In short, classification is a supervised task, whereas clustering is an unsupervised task
- There are many ways to perform clustering (just like there are many ways to fit regression or classification functions). We'll study the two of the most common techniques

2 K-means clustering

2.1 Within-cluster variation

- Suppose that we have n data points in p dimensions, $x_1, \dots x_n \in \mathbb{R}^p$. Let K be the number of clusters (consider this fixed). A *clustering* of the points $x_1, \dots x_n$ is a function C that assigns each observation x_i to a group $k \in \{1, \dots K\}$. Our notation: $C(i) = k$ means that x_i is assigned to group k, and n_k is the number of points in group k
- A natural objective is to choose the clustering C to minimize the within-cluster variation:

$$\sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2, \tag{1}$$

where $\bar{x}_k$ the average of points in group k,

$$\bar{x}_k = \frac{1}{n_k} \sum_{C(i)=k} x_i.$$

This is because the smaller the within-cluster variation, the tighter the clustering of points

- A problem is that doing this exactly is requires trying all possible assignments of the n points into K groups. The number of possible assignments is

$$N(n, K) = \frac{1}{K!} \sum_{k=1}^{K} (-1)^{K-k} \binom{K}{k} k^n.$$

Note that $N(10, 4) = 34,105$, and $A(25, 4) \approx 5 \times 10^{13}$ ... this is incredibly huge!

- Therefore we will have to settle for a clustering C that approximately minimizes the within-cluster variation. We will start by recalling the following fact: for any points $z_1, \dots z_m \in \mathbb{R}^p$, the quantity

$$\sum_{i=1}^{m} \|z_i - c\|_2^2$$

is minimized by choosing $c = \bar{z} = \frac{1}{m} \sum_{i=1}^{m} z_i$, the average of $z_1, \dots z_m$. Hence, minimizing (1) over a clustering assignment C is the same as minimizing the enlarged criterion

$$\sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - c_k\|_2^2 \tag{2}$$

over both the clustering C and centers $c_1, \dots c_K \in \mathbb{R}^p$

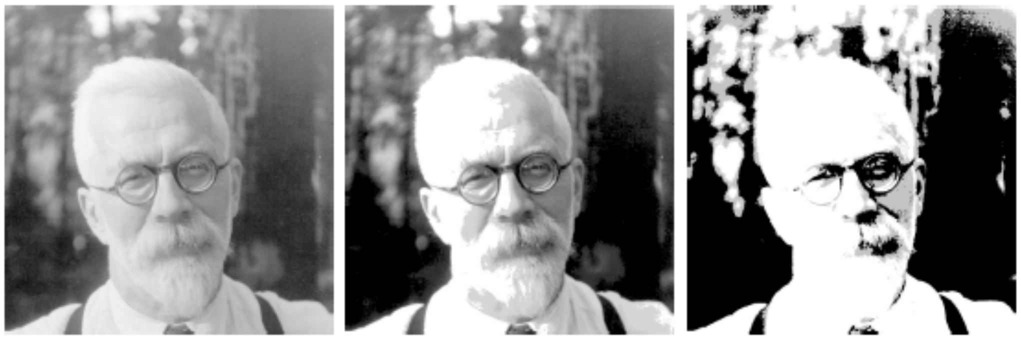
2.2 K-means clustering

- The *K-means clustering* algorithm approximately minimizes the enlarged criterion (2). It does so by alternately minimizing over the choice of clustering assignment C and the centers $c_1, \dots c_K$
- We start with an initial guess for $c_1, \dots c_K$ (e.g., pick K points at random over the range of $x_1, \dots x_n$), and then repeat:
    1. *Minimize over C*: for each $i = 1, \dots n$, find the cluster center c_k closest to x_i, and let $C(i) = k$
    2. *Minimize over $c_1, \dots c_K$*: for each $k = 1, \dots K$, let $c_k = \bar{x}_k$, the average of group k points

  We stop when the within-cluster variation doesn't change
- Put in words, this algorithm repeats two steps:
    1. Cluster (label) each point based the closest center
    2. Replace each center by the average of points in its clusterx
- Note that the within-cluster variation decreases with each iteration of the algorithm. I.e., if W_t is the within-cluster variation at iteration t, then $W_{t+1} \leq W_t$. Also, the algorithm always terminates, no matter the initial cluster centers. In fact, it takes $\leq K^n$ iterations (why?)
- The final clustering assignment generally depends on the initial cluster centers. Sometimes, different initial centers lead to very different final outputs. Hence we typically run K-means multiple times (e.g., 10 times), randomly initializing cluster centers for each run, then choose among from collection of centers based on which one gives the smallest within-cluster variation
- This is not guaranteed to deliver the clustering assignment that globally minimizes the within-cluster variation (recall: this would require looking through all possible assignments!), but it does tend to give us a pretty good approximation

2.3 Vector quantization

- K-means is often called "Lloyd's algorithm" in computer science and engineering, and is used in *vector quantization* for compression
- An example from the ESL book, page 514 is copied below.



  Left: original image; middle: using 23.9% of the storage; right: using 6.25% of the storage
- How did we do this? The basic idea: we run K-means clustering on 4×4 squares of pixels in an image, and keep only the clusters and labels. Smaller K means more compression. Then we can simply reconstruct the images at the end based on the clusters and the labels

2.4 K-medoids clustering

- A cluster center is a representative for all points in a cluster (and is also called a prototype). In K-means, we simply take a cluster center to be the average of points in the cluster. This is great for computational purposes—but how does it lend to interpretation?
- This would be fine if we were clustering, e.g., houses in Pittsburgh based on features like price, square footage, number of bedrooms, distance to nearest bus stop, etc. But not so if, e.g., we were clustering images of faces (why?)
- In some applications we want each cluster center to be one of the points itself. This is where *K-medoids clustering* comes in—it is a similar algorithm to the K-means algorithm, except when fitting the centers $c_1, \ldots c_K$, we restrict our attention to the points themselves
- As before, we start with an initial guess for the centers $c_1, \ldots c_K$ (e.g., randomly select K of the points $x_1, \ldots x_n$), and then repeat:

  1. *Minimize over C*: for each $i = 1, \ldots n$, find the cluster center c_k closest to x_i, and let $C(i) = k$
  2. *Minimize over $c_1, \ldots c_K$*: for each $k = 1, \ldots K$, let $c_k = x_k^*$, the *medoid* of the points in cluster k, i.e., the point x_i in cluster k that minimizes $\sum_{C(j)=k} \|x_j - x_i\|_2^2$

  We stop when the within-cluster variation doesn't change
- Put in words, this algorithm repeats two steps:

  1. Cluster (label) each point based on the closest center
  2. Replace each center by the medoid of points in its cluster
- The K-medoids algorithm shares the properties of K-means that we discussed: each iteration decreases the criterion; the algorithm always terminates; different starts gives different final answers; it does not achieve the global minimum
- Importantly, K-medoids generally returns a higher value of the criterion

  $$\sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - c_k\|_2^2$$

  than does K-means (why?). Also, K-medoids is computationally more challenging that K-means, because of its second repeated step: computing the medoid of points is harder than computing the average of points
- However, remember, K-medoids has the (potentially important) property that the centers are located among the data points themselves

3 Hierarchical clustering

3.1 Motivation from K-means

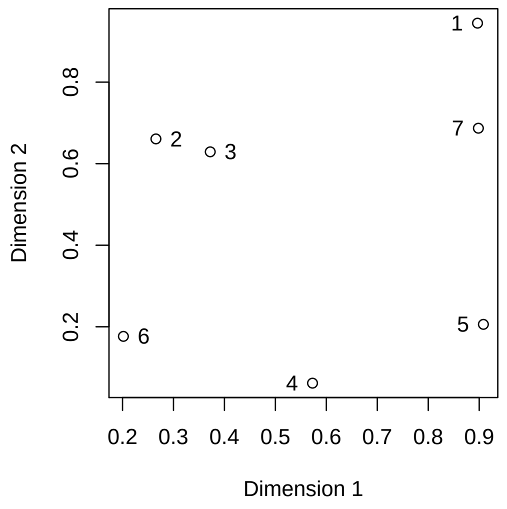
- Two properties of K-means (or K-medoids) clustering: (1) the algorithm fits exactly K clusters (as specified); (2) the final clustering assignment depends on the chosen initial cluster centers
- An alternative approach called *hierarchical clustering* produces a consistent result, without the need to choose initial starting positions (number of clusters). It also fits a sequence of clustering assignments, one for each possible number of underlying clusters $K = 1, \ldots n$

- The caveat: we need to choose a way to measure the dissimilarity between groups of points, called the linkage
- Given the linkage, hierarchical clustering produces a sequence of clustering assignments. At one end, all points are in their own cluster, at the other end, all points are in one cluster
- There are two types of hierarchical clustering algorithms: agglomerative, and divisive. In agglomerative, or bottom-up hierarhical clustering, the procedure is as follows:
  - Start with all points in their own group
  - Until there is only one cluster, repeatedly: merge the two groups that have the smallest dissimilarity

  Meanwhile, in divisive, or top-down hierarchical clustering, the procedure is as follows:
  - Start with all points in one cluster
  - Until all points are in their own cluster, repeatedly: split the group into two resulting in the biggest dissimilarity
- Agglomerative strategies are generally simpler than divisive ones, so we'll focus on them. Divisive methods are still important, but we won't be able to cover them due to time constraints

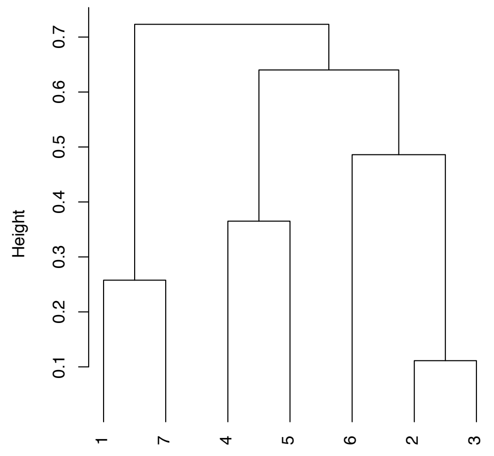
3.2 Dendrograms

- To understand the need for dendrograms, it helps to consider an simple example. Given the data points on the left, an agglomerative hierarhical clustering algorithm might decide on the clustering sequence described on the right



Step 1: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}$;
Step 2: $\{1\}, \{2, 3\}, \{4\}, \{5\}, \{6\}, \{7\}$;
Step 3: $\{1, 7\}, \{2, 3\}, \{4\}, \{5\}, \{6\}$;
Step 4: $\{1, 7\}, \{2, 3\}, \{4, 5\}, \{6\}$;
Step 5: $\{1, 7\}, \{2, 3, 6\}, \{4, 5\}$;
Step 6: $\{1, 7\}, \{2, 3, 4, 5, 6\}$;
Step 7: $\{1, 2, 3, 4, 5, 6, 7\}$.

- This is a cumbersome representation for the sequence of clustering assignments. Fortunately, we can also represent the sequence of clustering assignments using what is called a *dendrogram*:



Note that cutting the dendrogram horizontally partitions the data points into clusters

- In general, a dendrogram is a convenient graphic to display a hierarchical sequence of clustering assignments. It is simply a tree where:
  - Each node represents a group
  - Each leaf node is a singleton (i.e., a group containing a single data point)
  - Root node is the group containing the whole data set
  - Each internal node has two daughter nodes (children), representing the the groups that were merged to form it
- Remember: the choice of linkage determines how we measure dissimilarity between groups of points
- If we fix the leaf nodes at height zero, then each internal node is drawn at a height proportional to the dissmilarity between its two daughter nodes

3.3 Linkages

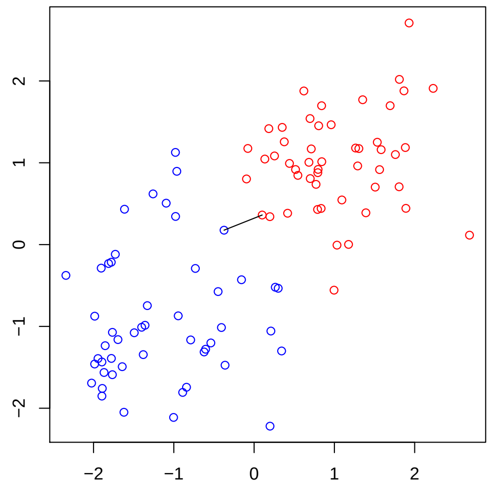
- Suppose that we are given points $x_1, \dots x_n$, and *dissimilarities* d_{ij} between each pair x_i and x_j. As an example, you may think of $x_i \in \mathbb{R}^p$ and $d_{ij} = \|x_i - x_j\|_2$
- At any stage of the hierarhical clustering procedure, clustering assignments can be expressed by sets $G = \{i_1, i_2, \dots i_r\}$, giving indices of points in this group. Let n_G be the size of G (here $n_G = r$). Bottom level: each group looks like $G = \{i\}$, top level: only one group, $G = \{1, \dots n\}$
- A *linkage* is a function $d(G, H)$ that takes two groups G, H and returns a dissimilarity score between them. Given the linkage, we can summarize agglomerative hierarhical clustering as follows:
  - Start with all points in their own group
  - Until there is only one cluster, repeatedly: merge the two groups G, H such that $d(G, H)$ is smallest

3.3.1 Single linkage

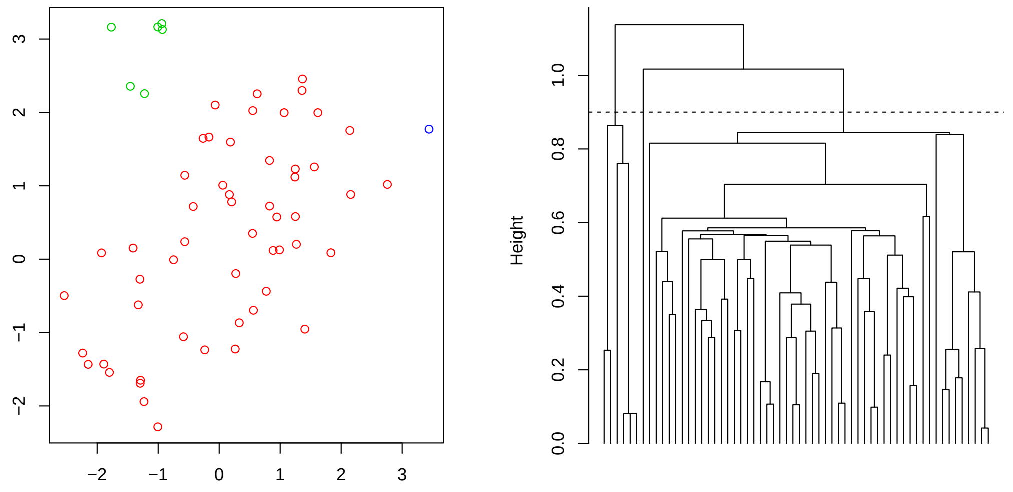
- In *single linkage* (i.e., nearest-neighbor linkage), the dissimilarity between G, H is the smallest dissimilarity between two points in opposite groups:

$$d_{\text{single}}(G,H) = \min_{i \in G,\, j \in H} d_{ij}$$

An illustration (dissimilarities d_{ij} are distances, groups are marked by colors): single linkage score $d_{\text{single}}(G,H)$ is the distance of the closest pair



- A single linkage clustering example: here $n = 60$, $x_i \in \mathbb{R}^2$, $d_{ij} = \|x_i - x_j\|_2$. Cutting the tree at $h = 0.9$ gives the clustering assignments marked by colors



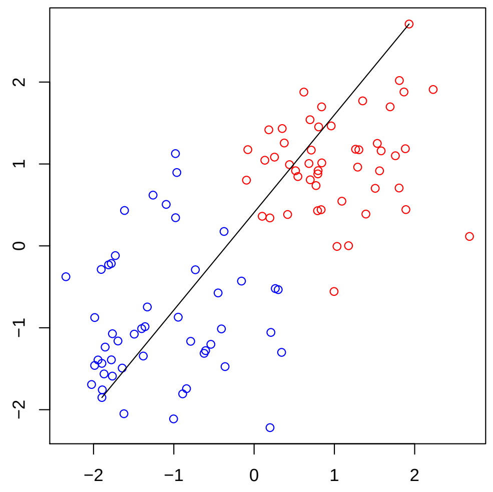
Cut interpretation: for each point x_i, there is another point x_j in its cluster with $d_{ij} \le 0.9$

3.3.2 Complete linkage

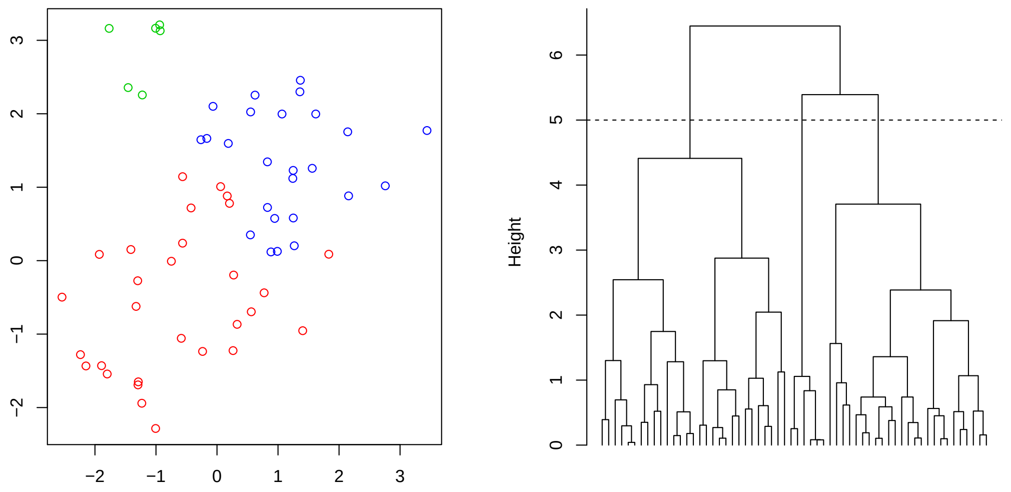
- In *complete linkage* (i.e., furthest-neighbor linkage), the dissimilarity between G, H is the largest dissimilarity between two points in opposite groups:

$$d_{\text{complete}}(G,H) = \max_{i \in G,\, j \in H} d_{ij}$$

An illustration (dissimilarities d_{ij} are distances, groups are marked by colors): complete linkage score $d_{\text{complete}}(G, H)$ is the distance of the furthest pair



- A complete linkage clustering example: same data set as before. Cutting the tree at $h = 5$ gives the clustering assignments marked by colors



Cut interpretation: for each point x_i, every other point x_j in its cluster satisfies $d_{ij} \leq 5$

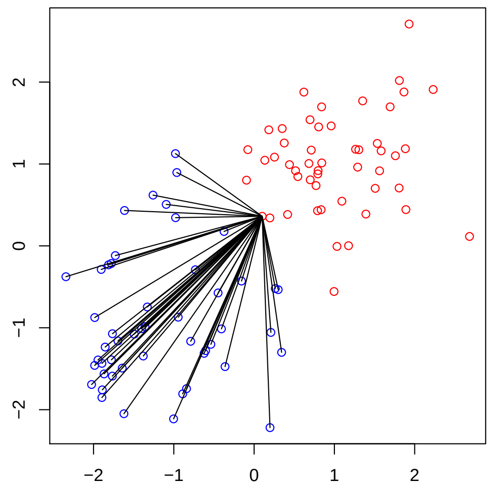
3.3.3 Average linkage

- In *average linkage*, the dissimilarity between G, H is the average dissimilarity over all points in opposite groups:

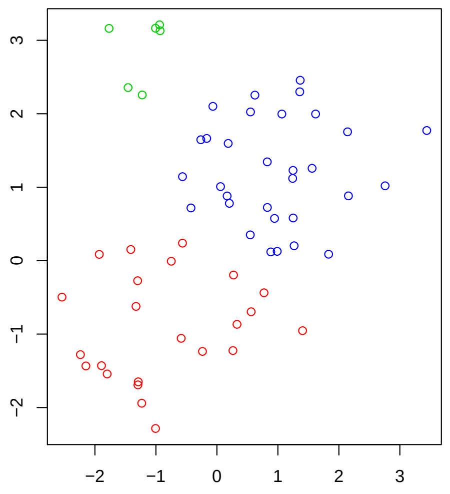
$$d_{\text{average}}(G, H) = \frac{1}{n_G \cdot n_H} \sum_{i \in G,\, j \in H} d_{ij}$$

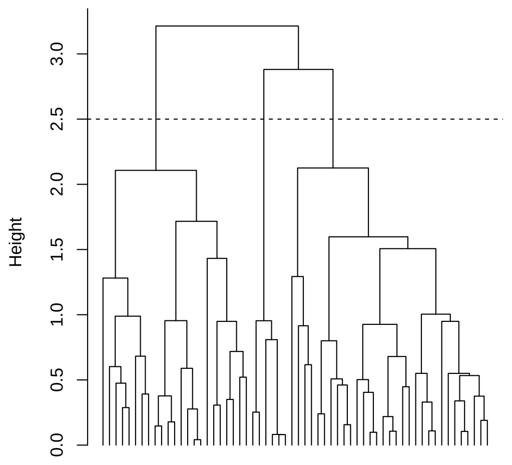
An illustration (dissimilarities d_{ij} are distances, groups are marked by colors): average linkage score $d_{\text{average}}(G, H)$ is the average distance across all pairs

(Plot here only shows distances between the blue points and one red point)



- An average linkage clustering example: same data set as before. Cutting the tree at $h = 1.5$ gives clustering assignments marked by the colors





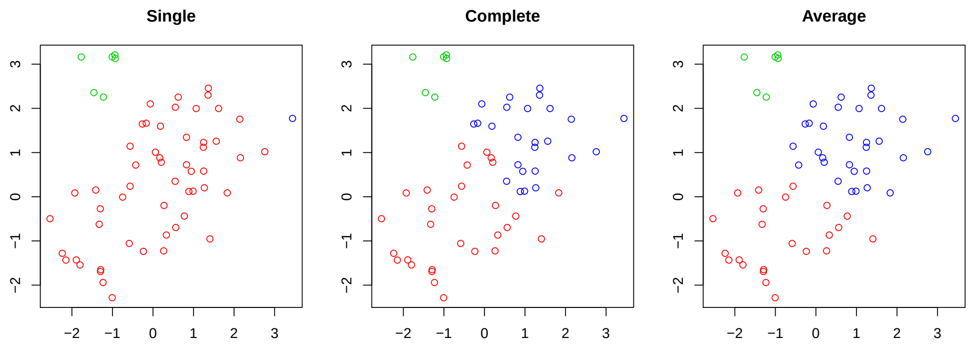
Cut interpretation: there really isn't a good one!

3.3.4 Common properties of linkages

- Single, complete, average linkage share some nice properties
- These linkages operate on dissimilarities d_{ij}, and don't need the data points $x_1, \ldots x_n$ to be in Euclidean space
- Also, running agglomerative clustering with any of these linkages produces a dendrogram with *no inversions*
- This last property, in words: dissimilarity scores between merged clusers only increases as we run the algorithm. It means that we can draw a proper dendrogram, where the height of a parent is always higher than height of its daughters

3.3.5 Shortcomings of single, complete linkage

- Single and complete linkage can have some practical problems
- Single linkage suffers from an issue we call *chaining*. In order to merge two groups, only need one pair of points to be close, irrespective of all others. Therefore clusters can be too spread out, and not compact enough
- Complete linkage avoids chaining, but suffers from an issue called *crowding*. Because its score is based on the worst-case dissimilarity between pairs, a point can be closer to points in other clusters than to points in its own cluster. Clusters are compact, but not far enough apart
- Average linkage tries to strike a balance. It uses average pairwise dissimilarity, so clusters tend to be relatively compact and relatively far apart
- Recall that we've already seen examples of chaining and crowding:



3.3.6 Shortcomings of average linkage

- Average linkage isn't perfect, and it has its own problems
- It is not clear what properties the resulting clusters have when we cut an average linkage tree at given height h. Single and complete linkage trees each had simple interpretations
- Moreover, the results of average linkage clustering can change with a monotone increasing transformation of dissimilarities d_{ij}. I.e., if h is such that $h(x) \leq h(y)$ whenever $x \leq y$, and we used dissimilarites $h(d_{ij})$ instead of d_{ij}, then we could get different answers
- Depending on the context, this last problem may be important or unimportant. E.g., it could be very clear what dissimilarities should be used, or not
- Note: the results of single, complete linkage clustering are unchanged under monotone transformations

4 Choosing the number of clusters

4.1 An important, hard problem

- Sometimes, when using K-means, K-medoids, or hierarchical clustering, we might have no problem specifying the number of clusters K ahead of time, e.g., segmenting a client database into K clusters for K salesman; or, compressing an image using vector quantization, where K controls the compression rate

- Other times, K is implicitly defined by cutting a hierarchical clustering tree at a given height, determined by the application

- But in most exploratory applications, the number of clusters K is unknown. So we are left asking the question: what is the "right" value of K?

- This is a very hard problem! Why? Determining the number of clusters is a hard task for humans to perform (unless the data are low-dimensional). Not only that, it's just as hard to explain what it is we're looking for. Usually, statistical learning is successful when at least one of these is possible

- Why is it important? It can make a big difference for the interpretation of the clustering result in the application, e.g., it might make a big difference scientifically if we were convinced that there were $K = 2$ subtypes of breast cancer vs. $K = 3$ subtypes. Also, one of the (larger) goals of data mining/statistical learning is automatic inference; choosing K is certainly part of this

- For concreteness, we're going to focus on K-means, but most ideas will carry over to other settings. Recall that, given the number of clusters K, the K-means algorithm approximately minimizes the within-cluster variation:

$$W_K = \sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2$$

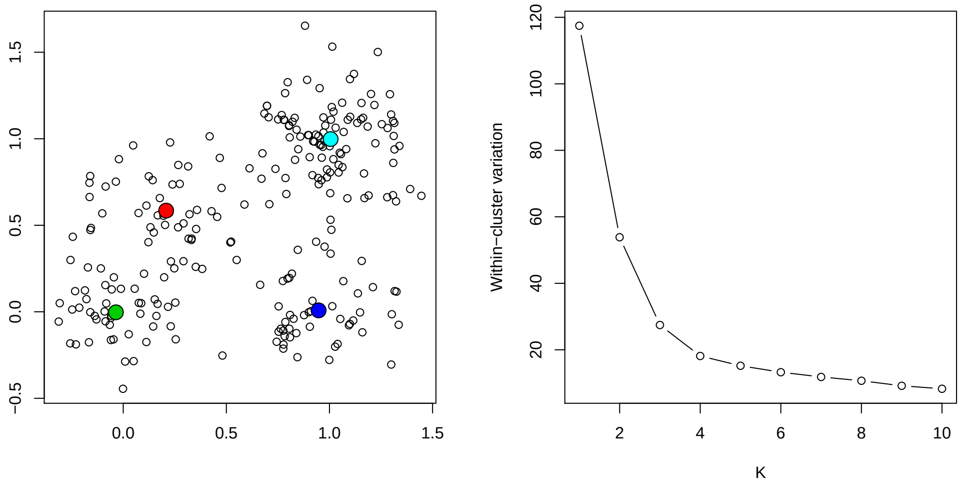
  over cluster assignments C, where $\bar{x}_k$ is the average of points in group k, $\bar{x}_k = \frac{1}{n_k}\sum_{C(i)=k} x_i$. We'll start off discussing a method for choosing K that doesn't work, and then discuss two methods can work well, in some cases

4.2 That's not going to work!

- Clearly a lower value of W_K is better. So why not just run K-means for a bunch of different values of K, and choose the value of K that gives the smallest W_K?

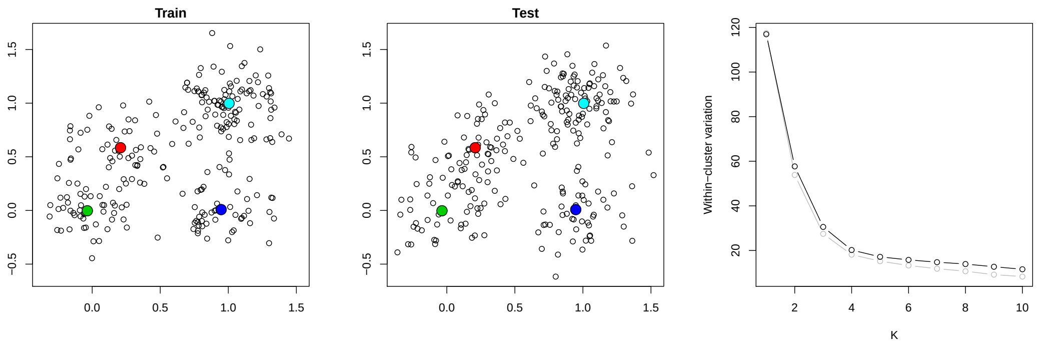
- That's not going to work! The problem: within-cluster variation W_K just keeps decreasing as K increases

- Here's an example: $n = 250$, $p = 2$, $K = 1, \ldots 10$. Below we plot the data points on the left, and the within-cluster variation as a function on the right



This behavior shouldn't be surprising, once we think about K-means clustering and the definition of within-cluster variation

- Interestingly, even if we had computed the centers on training data, and evaluated W_K on test data (assuming we had it), then there would still a problem! Same example as above, but now the within-cluster variation on the right has been computed on the test data set:



4.3 The CH index

- Within-cluster variation measures how tightly grouped the clusters are. As we increase the number of clusters K, this just keeps going down. What are we missing?

- *Between-cluster variation* measures how spread apart the groups are from each other:

$$B_K = \sum_{k=1}^{K} n_k \|\bar{x}_k - \bar{x}\|_2^2$$

where as before $\bar{x}_k$ is the average of points in group k, and $\bar{x}$ is the overall average, i.e.

$$\bar{x}_k = \frac{1}{n_k} \sum_{C(i)=k} x_i \quad \text{and} \quad \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- Bigger B_K is better, so can we use it to choose K? It's still not going to work alone, for a similar reason to the one above: between-cluster variation just increases as K increases

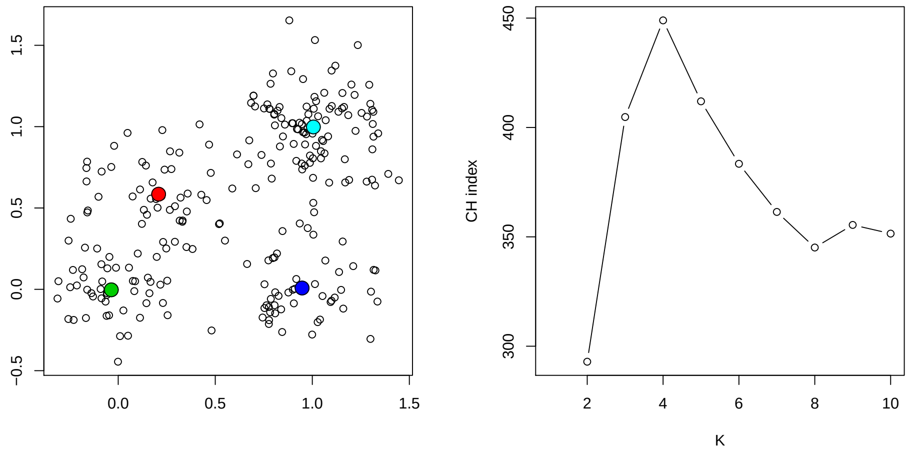
- A helpful realization: ideally we'd like our clustering assignments C to simultaneously have a small W_K and a large B_K. This is the idea behind the *CH index* (named after the two statisticians who proposed it, Calinski and Harabasz). For clustering assignments coming from K clusters, we record the CH score:

$$\text{CH}_K = \frac{B_K/(K-1)}{W_K/(n-K)}.$$

To choose K, we just pick some maximum number of clusters to be considered $K_{\max}$ (e.g., $K = 20$), and choose the value of K with the largest score CH_K,

$$\hat{K} = \underset{K \in \{2, \ldots K_{\max}\}}{\operatorname{argmax}} \; \text{CH}_K$$

- On our running example ($n = 250$, $p = 2$, $K = 2, \ldots 10$):



We would choose $K = 4$ clusters, which seems reasonable

- A general problem: the CH index is not defined for $K = 1$. So, using it, we could never choose to fit just one cluster (the null model)!

4.4 The gap statistic

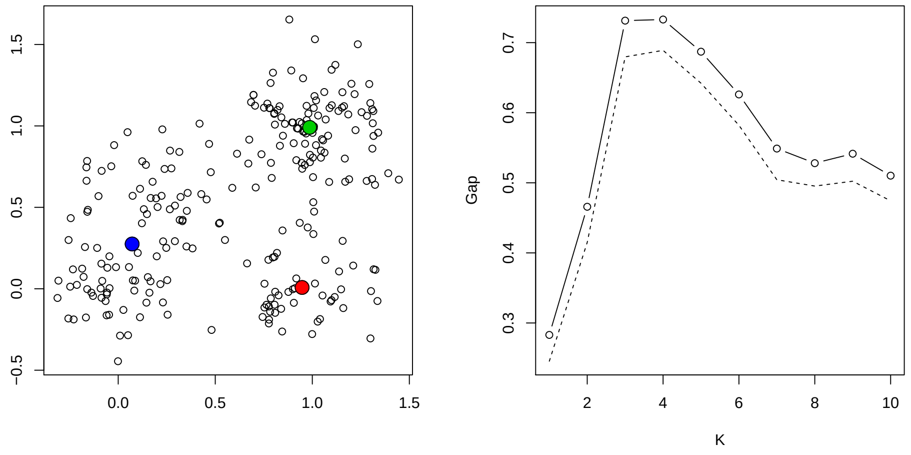
- It's true that W_K keeps dropping, but how much it drops at as we progress from K to $K+1$ should be informative. The *gap statistic* is based on this idea.[1] We compare the observed within-cluster variation W_K to W_K^{unif}, the within-cluster variation we'd see if we instead had points distributed uniformly (over an encapsulating box). The gap for K clusters is defined as

$$\text{Gap}_K = \log W_K - \log W_K^{\text{unif}}$$

- The quantity $\log W_K^{\text{unif}}$ is computed by simulation: we average the log within-cluster variation over, say, 20 simulated uniform data sets. We also compute the standard deviation of s_K of $\log W_K^{\text{unif}}$ over the simulations. Then we choose K by

$$\hat{K} = \min\Big\{K \in \{1, \dots K_{\max}\} : \text{Gap}_K \geq \text{Gap}_{K+1} - s_{K+1}\Big\}$$

- On our running example ($n = 250$, $p = 2$, $K = 1, \dots 10$):



[1]Tibshirani et al. (2001), "Estimating the number of clusters in a data set via the gap statistic"

Here we would choose $K = 3$ clusters, which is also reasonable

- The gap statistic does especially well when the data fall into one cluster. Why? (Hint: think about the null distribution that it uses)