Chapter 19

Survival Trees

19.1 The Log-Rank Test

The **log-rank test** (Mantel 1966; Peto and Peto 1972) is a test for statistical equivalence of two survival curves. It is obtained by constructing a $2x2$ contingency table at the time of each event and comparing the failure rates between the two groups, conditional on the number at risk in each group[1]. In this way, the test compares the entire survival experience between groups. The null hypothesis is that the true underlying curves for the two groups are identical.

"In the absence of censoring, these methods reduce to the Wilcoxon-Mann-Whitney rank-sum test (Mann and Whitney 1947) for two samples and to the Kruskal-Wallis test (Kruskal and Wallis 1952) for more than two groups of survival times."

19.2 Survival Example: Primary Biliary Cirrhosis

Now let's consider how the same tree-building machinery

This data is from the Mayo Clinic trial in primary biliary cirrhosis (PBC) of the liver conducted between 1974 and 1984. A total of 424 PBC patients,

[1]See https://bookdown.org/sestelo/sa_financial/comparing-survival-curves.html.

referred to Mayo Clinic during that ten-year interval, met eligibility criteria for the randomized placebo controlled trial of the drug D-penicillamine. The first 312 cases in the data set participated in the randomized trial and contain largely complete data. The additional 112 cases did not participate in the clinical trial, but consented to have basic measurements recorded and to be followed for survival. Six of those cases were lost to follow-up shortly after diagnosis, so the data here are on an additional 106 cases as well as the 312 randomized participants.

Question 19.1

Here's a dataset of survival data... how would you build a survival forest?

19.3 Random Survival Forests

19.4 Boosted Survival Trees

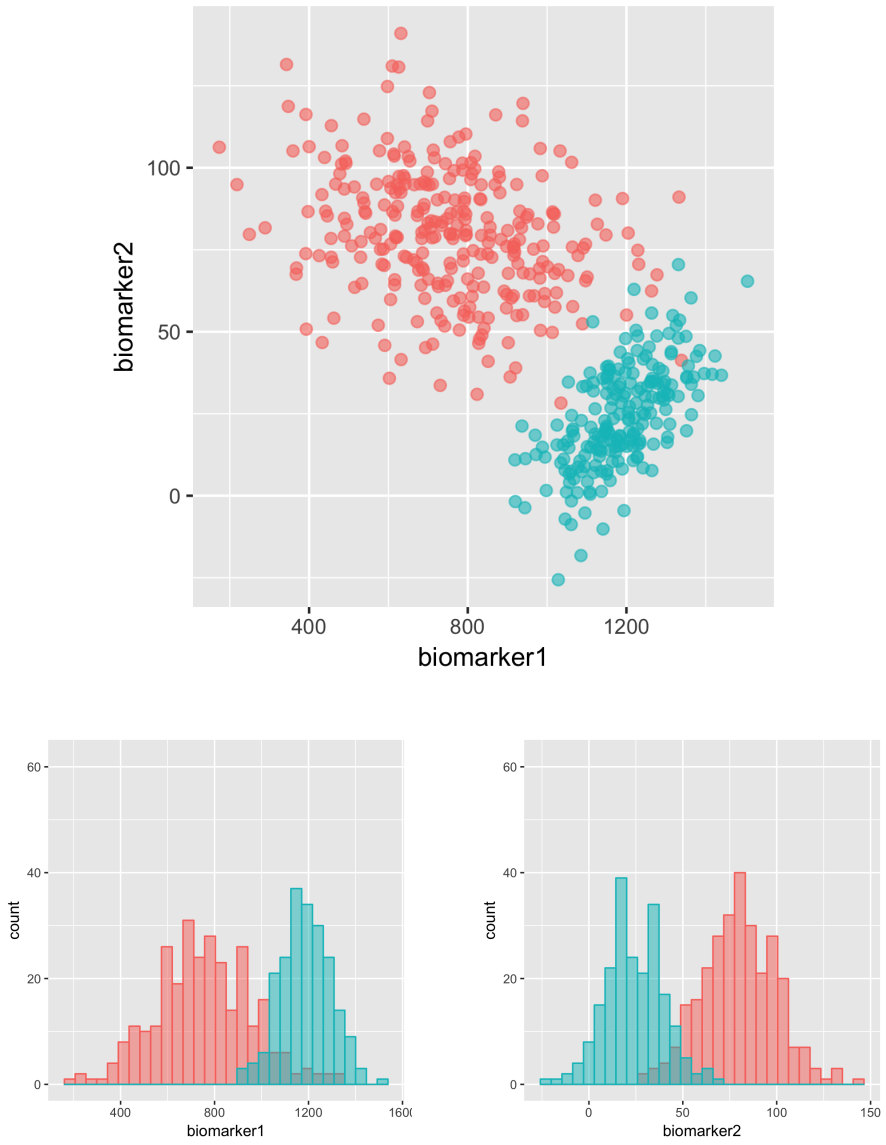
Chapter 20

Clustering

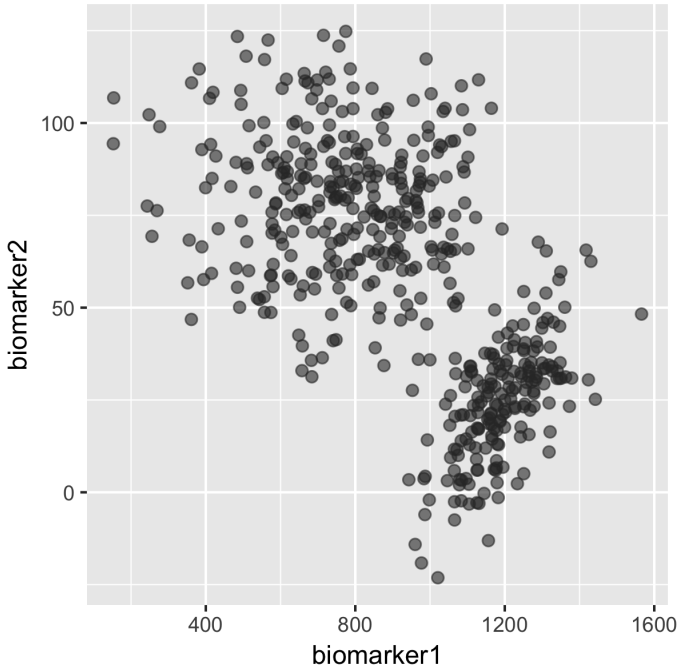
Our discussions so far have focused on the problem of supervised learning, in which we create a mapping between a set of inputs and an output. In this chapter, we turn our attention to the problem of **unsupervised learning**, in which our goal is to uncover hidden structure in a dataset. There is no "special" outcome variable in these types of problems. Way back in Chapter 1, examples 7, 8, and 14 were examples of unsupervised learning problems.

20.1 A Thought Experiment: Flow Cytometry Data

Imagine you have data on 500 cells from two different cell lines. For each cell, you record the fluorescence intensity of two different biomarkers, biomarker 1 (x_1) and biomarker 2 (x_2). The data are shown below.



In real life, you would not have the labels for the two cell lines. You would have no idea which distribution(s) the data were drawn from or what the probabilities associated with the various distributions were. Real flow cytometry data would instead look like this:



The human eye can distinguish structure in a plot like this, but it's harder to train a computer to see it, and it's even harder to prove that the structure the computer finds is real. Over the years, people have tried many different strategies.

Question 20.1

Speculate on some possible approaches for separating data that look like this into groups, or clusters.

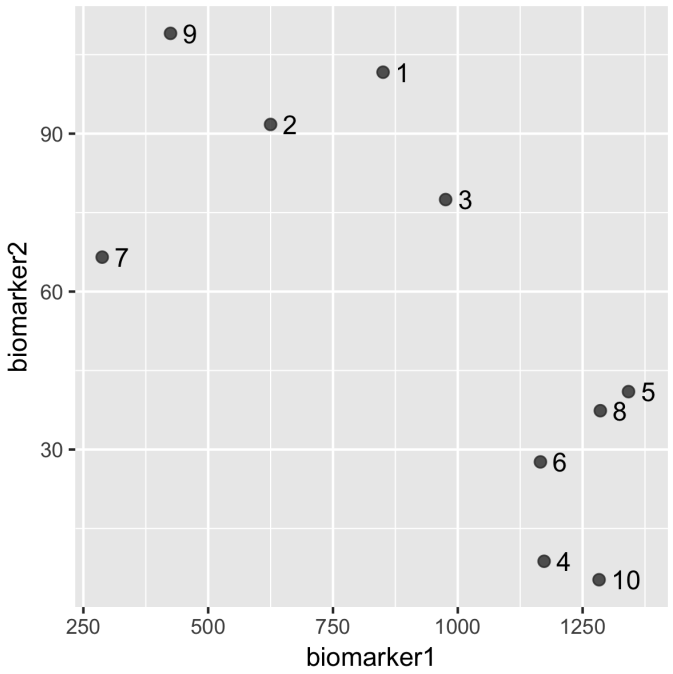
20.1.1 Downsampled Dataset

For the rest of this workshop, we need to be able to calculate things by hand, so we will downsample the flow cytometry data, above, to just 10 datapoints. The downsampled, labeled data are shown here:

i	Biomarker 1 Intensity $(x_1^{(i)})$	Biomarker 2 Intensity $(x_2^{(i)})$	Cell Line (z)
1	634.83	110.55	B
2	650.06	74.22	B
3	788.24	81.52	B
4	771.47	84.98	B
5	515.81	91.08	B
6	1101.23	31.05	A
7	649.32	77.05	B
8	652.89	97.16	B
9	1183.02	11.73	A
10	1238.45	33.46	A

Without their labels, the data look like this:

i	Biomarker 1 Intensity $(x_1^{(i)})$	Biomarker 2 Intensity $(x_2^{(i)})$	Cell Line (z)
1	634.83	110.55	
2	650.06	74.22	
3	788.24	81.52	
4	771.47	84.98	
5	515.81	91.08	
6	1101.23	31.05	
7	649.32	77.05	
8	652.89	97.16	
9	1183.02	11.73	
10	1238.45	33.46	



20.2 K-Means

Let's first consider a very simple unsupervised machine learning algorithm called **K-means**. The goal of K-means is to cluster [unlabeled] data into groups so that the distances between points within a group are minimized.

Assume you have a dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, where each vector $x^{(i)}$ is of length p, and you want to cluster these n vectors into K distinct groups. Here is the K-means algorithm:

1. Assign each of the n datapoints to a random cluster. You can do this one of three ways:

   (1) Choose a random cluster for each point independently.

   (2) Choose K initial points to be the cluster centers.

   (3) Choose K initial points uniformly within the feature space (not necessarily data point locations) to be the cluster centers.

After the initial cluster assignments are made, proceed to the update step, below.

2. **Assignment step.** Assign each point to the cluster whose mean is the closest, using Euclidean distance. Mathematically:

$$c_i^{(t)} := \text{argmin}_j \|x^{(i)} - \mu_j\|$$

where we note that the distance is given by the L_2, or Euclidean, norm.

3. **Update step.** Calculate the means to be the centroids of the points in the clusters.

$$\mu_j^{(t)} := \frac{\sum_{i=1}^{n} x^{(i)} \cdot \mathbb{I}\left\{c^{(i)} = j\right\}}{\sum_{i=1}^{n} \mathbb{I}\left\{c^{(i)} = j\right\}}$$

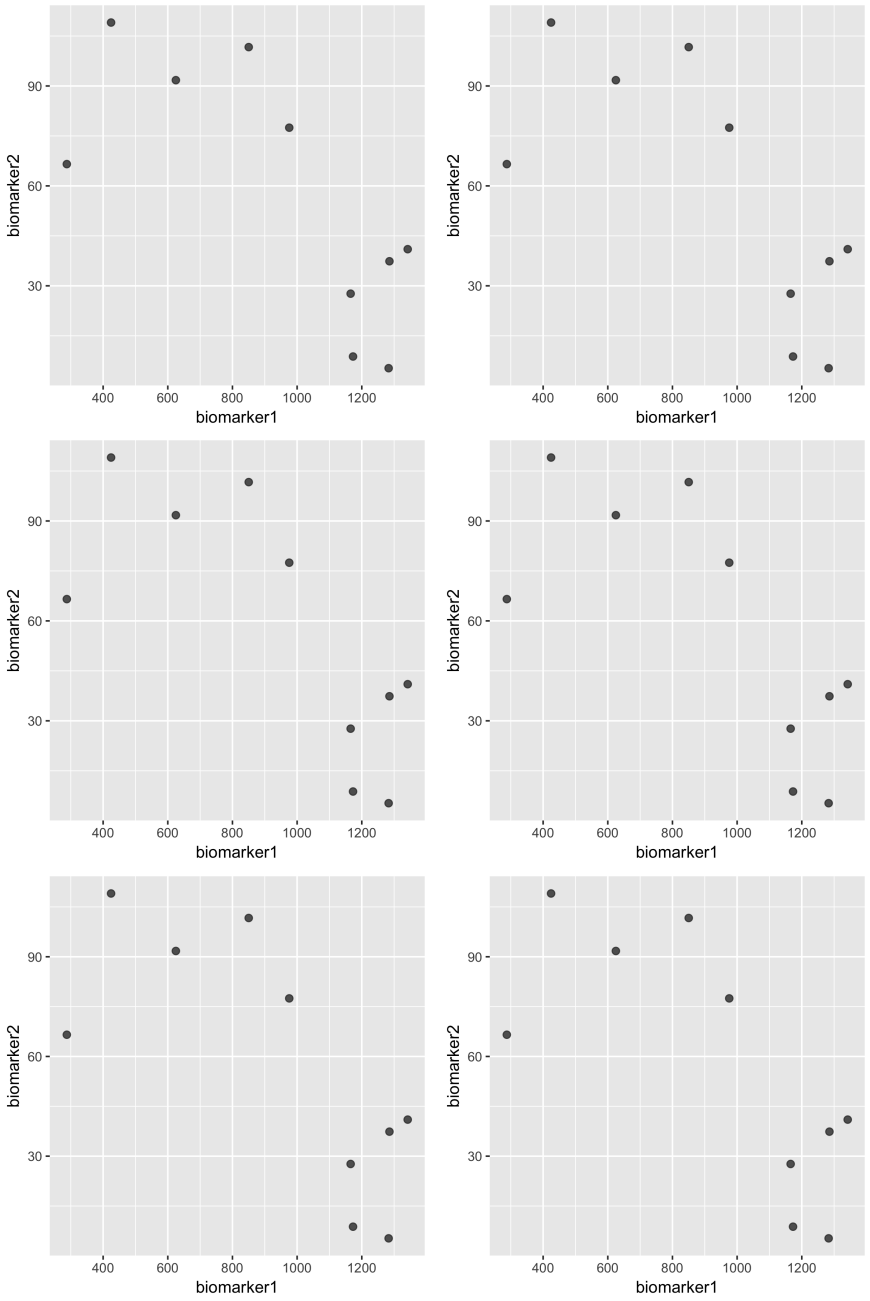
4. Repeat Steps 2 and 3 until no points change clusters (or until convergence).

Question 20.2

Which supervised learning algorithm does this remind you of? What is different between K-means and this algorithm?

Question 20.3

Below is our unlabeled, downsampled flow cytometry dataset. Cluster it into two groups using K-means. You can initialize your clusters however you want. (Note: Assume that the data were standardized in advance so that the spacing between the white lines equals one "unit" for both biomarker 1 and biomarker 2.)



Question 20.4

What are some disadvantages of K-means?

Question 20.5

One application of K-means clustering is image compression, as shown in the figure below (which is from Christopher Bishop's classic book *Pattern Recognition and Machine Learning*).

Draw a picture of the data matrix for the image compression clustering problem. How will the clustering work here? Are we clustering rows or columns? What are the dimensions of the dataset, n and p?

20.3 Mixture Models

Mixture models represent data using mixtures of simple probability distributions. They are a step up in methodological rigor from K-means and are much more flexible, although the basic idea is the same.

Here are the key similarities:

1. The number of clusters, K, is still an input parameter.
2. Mixture models can still converge to local minima depending on the initialization.

Here are the key differences:

1. Mixture models use a **soft clustering** instead of a hard clustering. Each datapoint is assigned a probability distribution over potential clusters.

2. Instead of a distance metric like Euclidean distance, points are distributed over clusters by considering probability densities of the various clusters and how likely it is that the point came from each probability density.

3. No need to assume axes are on an equal scale; the individual probability distributions can account for this.

We'll examine mixtures of Gaussians today because they're easy to visualize, but in reality the mixture components can be any type of well-behaved probability distribution.

20.4 Multivariate Gaussian Distribution

We already saw the univariate Gaussian in Chapter 4. The **multivariate Gaussian** is an extension of the Gaussian to multiple dimensions.

20.4.1 Probability Density and Parameters

The m-dimensional **multivariate Gaussian** probability distribution is given by:

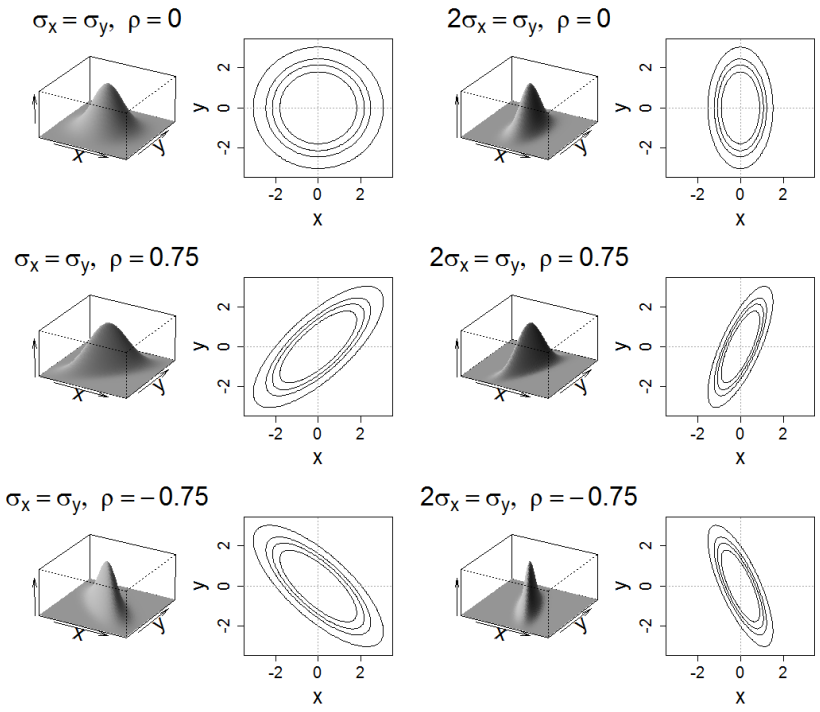
$$p(x|\mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^m |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right)$$

where $x \in \mathbb{R}^m$ (a vector of length m) and the mean, μ, is also a vector of length m. The variance of a univariate Gaussian, σ, is replaced by a covariance matrix of dimension $m \times m$:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \dots & \rho_{1m}\sigma_1\sigma_m \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{m1}\sigma_m\sigma_1 & \rho_{m2}\sigma_m\sigma_2 & \dots & \sigma_m^2 \end{bmatrix}$$

where ρ_{ij} is the Pearson correlation of X_i and X_j. Or alternatively, the ijth element of the covariance matrix is $\text{cov}(X_i, X_j) = E\left[(X_i - \mu_i)(X_j - \mu_j)\right]$.

In two dimensions, the cross-sections of the multivariate Gaussian distribution are ellipses. The axes of the ellipses are given by the eigenvectors of the covariance matrix, $\mathbf{\Sigma}$. The first and second eigenvalues of the covariance matrix give the variance of the data along the major and minor axes of the ellipses, respectively. Some examples of bivariate normal distributions are shown below. This figure and the code that generated it can be found here: `http://www.stat.cmu.edu/~kass/KEB/RHTML/R/bivariateNormalPerspectives.r.html`



20.4.2 Maximum Likelihood Estimates

To fit data to a multivariate Gaussian distribution, we once again use our old friend, maximum likelihood estimation (Chapter 5. Here are the maximum likelihood estimates for μ and $\mathbf{\Sigma}$ for the multivariate Gaussian:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)} \qquad \hat{\mathbf{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^T$$

20.5 Gaussian Mixture Models

A Gaussian mixture model fits a set of unlabeled data to a set of K multivariate Gaussians. There are three sets of parameters that the algorithm needs to identify:

1. $\mu_1, \ldots, \mu_K$ (the means of the Gaussians)
2. $\mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K$ (the covariance matrices of the Gaussians)
3. $\phi_1, \ldots, \phi_K$ (the mixing proportions, which must sum to one)

Question 20.6

Mixture models are examples of **generative models**, which tell a story about how the observed data were generated. Below is the actual code that generated the data for the flow cytometry example.

```
```{r Sampling from two cell lines}
cell_line_sample <- function(N) {
  mean.0 <- c(1200, 25)
  mean.1 <- c(750, 80)
  cov.0 <- matrix(c(100^2, 0.6*100*15, 0.6*100*15, 15^2), nrow=2)
  cov.1 <- matrix(c(200^2, -0.4*200*10, -0.4*200*10, 20^2), nrow=2)
  p.group.0 <- 0.4

  group <- runif(N)
  rand.samples = {}
  for(i in 1:N){
      if (group[i] < p.group.0) {
          rand.samples <- rbind(rand.samples, mvrnorm(1, mean.0, cov.0))
      } else {
          rand.samples <- rbind(rand.samples, mvrnorm(1, mean.1, cov.1))
      }
  }
  rand.samples <- as.data.frame(rand.samples)
  names(rand.samples) <- c("biomarker1", "biomarker2")
  rand.samples$group <- group < p.group.0
  rand.samples$name <- seq(1, N)
  return(rand.samples)
}
```
```

It turns out that this code matches the "story" of the Gaussian mixture model perfectly. (With real data, of course, this would not be the case.) What is that story?

Here is the algorithm for fitting a Gaussian mixture model:

1. Initialize the means μ_k, covariances $\mathbf{\Sigma}_k$, and mixing coefficients ϕ_k for all of the Gaussians $k = 1, \ldots, K$.

2. **E step.** Give each point a "voting weight" in each Gaussian equal to the probability (based on current parameter values) that it came from that Gaussian:

$$\begin{aligned} w_j^{(i)} &:= p(z^{(i)} = j | x^{(i)}, \phi, \mu, \mathbf{\Sigma}) \\ &= \frac{\phi_j \cdot \mathcal{N}(x^{(i)} | \mu_j, \mathbf{\Sigma}_j)}{\sum_{k=1}^{K} \phi_k \cdot \mathcal{N}(x^{(i)} | \mu_k, \mathbf{\Sigma}_k)} \end{aligned}$$

   Note that you will have K different voting weights for each point, and there are n points, so you need to do nK total calculations here.

3. **M step.** Re-estimate the parameters for the different Gaussians by letting each point vote in each Gaussian according to its voting weight.

$$\begin{aligned} \phi_j &:= \frac{1}{n} \sum_{i=1}^{n} w_j^{(i)} \\ \mu_j &:= \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}} \\ \mathbf{\Sigma}_j &:= \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)}} \end{aligned}$$

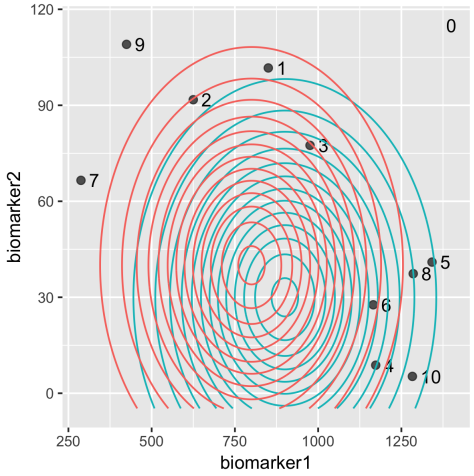
4. Check for convergence of either the parameters or the log-likelihood. If the convergence criterion is not satisfied, return to step 2.

Note that you are *not* guaranteed to get the right answer; the final Gaussians can change depending on how you initialize the model.

20.6 A Gaussian Mixture Model for Flow Cytometry Data

We will now follow the steps from the algorithm above to fit a Gaussian mixture model to the dataset from our flow cytometry example.

1. Initialize the means μ_A and μ_B, the covariances $\mathbf{\Sigma}_A$ and $\mathbf{\Sigma}_B$, and the mixing coefficients ϕ_A and ϕ_B.

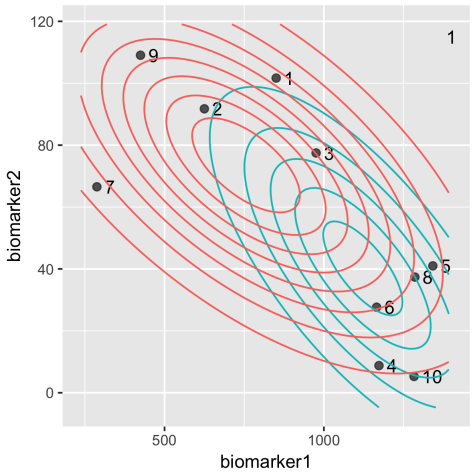


$$\phi_A = 0.5$$
$$\phi_B = 0.5$$
$$\mu_A = \begin{bmatrix} 900 \\ 30 \end{bmatrix}$$
$$\mu_B = \begin{bmatrix} 800 \\ 40 \end{bmatrix}$$
$$\mathbf{\Sigma}_A = \begin{bmatrix} 200^2 & 0 \\ 0 & 30^2 \end{bmatrix}$$
$$\mathbf{\Sigma}_B = \begin{bmatrix} 200^2 & 0 \\ 0 & 30^2 \end{bmatrix}$$

2. Do E step for round 1.

i	$x_1^{(i)}$	$x_2^{(i)}$	$\mathcal{N}(x^{(i)}\|\mu_A, \mathbf{\Sigma}_A)$	$\mathcal{N}(x^{(i)}\|\mu_B, \mathbf{\Sigma}_B)$	$w_A^{(i)}$	$w_B^{(i)}$
1	634.83	110.55	3e-07	1.2e-06	0.201	0.799
2	650.06	74.22	4.1e-06	1e-05	0.282	0.718
3	788.24	81.52	5.2e-06	1e-05	0.338	0.662
4	771.47	84.98	4e-06	8.5e-06	0.320	0.680
5	515.81	91.08	5.3e-07	2.3e-06	0.189	0.811
6	1101.23	31.05	1.6e-05	8.2e-06	0.662	0.338
7	649.32	77.05	3.5e-06	9.3e-06	0.275	0.725
8	652.89	97.16	1e-06	3.3e-06	0.234	0.766
9	1183.02	11.73	8.1e-06	2.7e-06	0.749	0.251
10	1238.45	33.46	6.3e-06	2.3e-06	0.729	0.271
sum					3.979	6.021

3. Do M step for round 1.



$$\phi_A = 0.398$$

$$\phi_B = 0.602$$

$$\mu_A = \begin{bmatrix} 947.6 \\ 53.5 \end{bmatrix}$$

$$\mu_B = \begin{bmatrix} 733.2 \\ 79.7 \end{bmatrix}$$

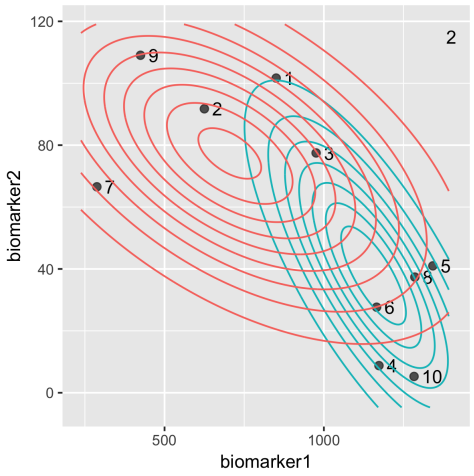
$$\Sigma_A = \begin{bmatrix} 256.6^2 & -0.925 \cdot 256.6 \cdot 32.3 \\ -0.925 \cdot 256.6 \cdot 32.3 & 32.3^2 \end{bmatrix}$$

$$\Sigma_B = \begin{bmatrix} 195.4^2 & -0.855 \cdot 195.4 \cdot 24.7 \\ -0.855 \cdot 195.4 \cdot 24.7 & 24.7^2 \end{bmatrix}$$

4. Do E step for round 2.

i	$x_1^{(i)}$	$x_2^{(i)}$	$\mathcal{N}(x^{(i)}\|\mu_A, \Sigma_A)$	$\mathcal{N}(x^{(i)}\|\mu_B, \Sigma_B)$	$w_A^{(i)}$	$w_B^{(i)}$
1	634.83	110.55	5.9e-06	1.6e-05	0.193	0.807
2	650.06	74.22	1.4e-05	3.1e-05	0.226	0.774
3	788.24	81.52	3.1e-05	5.1e-05	0.287	0.713
4	771.47	84.98	2.7e-05	4.8e-05	0.271	0.729
5	515.81	91.08	7.2e-06	2.2e-05	0.178	0.822
6	1101.23	31.05	3.9e-05	8.5e-06	0.754	0.246
7	649.32	77.05	1.7e-05	3.8e-05	0.227	0.773
8	652.89	97.16	2e-05	4.6e-05	0.219	0.781
9	1183.02	11.73	1.7e-05	1.5e-06	0.884	0.116
10	1238.45	33.46	1.4e-05	1.8e-06	0.837	0.163
sum					4.078	5.922

5. Do M step for round 2. Calculate the new log-likelihood.



$$\phi_A = 0.408$$

$$\phi_B = 0.592$$

$$\mu_A = \begin{bmatrix} 981.2 \\ 49.4 \end{bmatrix}$$

$$\mu_B = \begin{bmatrix} 706.5 \\ 83 \end{bmatrix}$$

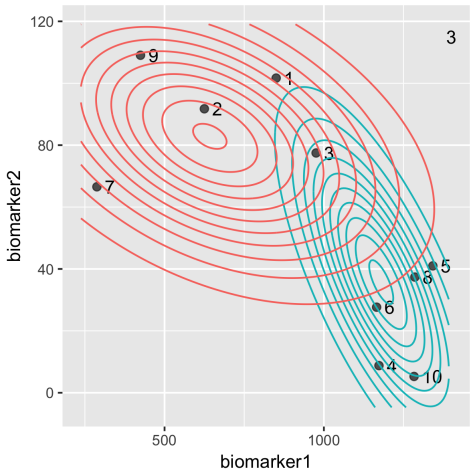
$$\Sigma_A = \begin{bmatrix} 252.6^2 & -0.924 \cdot 252.6 \cdot 32.1 \\ -0.924 \cdot 252.6 \cdot 32.1 & 32.1^2 \end{bmatrix}$$

$$\Sigma_B = \begin{bmatrix} 164.3^2 & -0.793 \cdot 164.3 \cdot 20.8 \\ -0.793 \cdot 164.3 \cdot 20.8 & 20.8^2 \end{bmatrix}$$

6. Do E step for round 3.

i	$x_1^{(i)}$	$x_2^{(i)}$	$\mathcal{N}(x^{(i)}\|\mu_A, \Sigma_A)$	$\mathcal{N}(x^{(i)}\|\mu_B, \Sigma_B)$	$w_A^{(i)}$	$w_B^{(i)}$
1	634.83	110.55	5e-06	1.9e-05	0.153	0.847
2	650.06	74.22	1.1e-05	3.8e-05	0.171	0.829
3	788.24	81.52	2.8e-05	5.9e-05	0.250	0.750
4	771.47	84.98	2.4e-05	5.6e-05	0.230	0.770
5	515.81	91.08	5.4e-06	2.7e-05	0.122	0.878
6	1101.23	31.05	4.3e-05	2.7e-06	0.917	0.083
7	649.32	77.05	1.4e-05	4.7e-05	0.171	0.829
8	652.89	97.16	1.7e-05	5.7e-05	0.167	0.833
9	1183.02	11.73	2e-05	2.1e-07	0.985	0.015
10	1238.45	33.46	1.6e-05	3.8e-07	0.965	0.035
sum					4.132	5.868

7. Do M step for round 3.



$$\phi_A = 0.413$$

$$\phi_B = 0.587$$

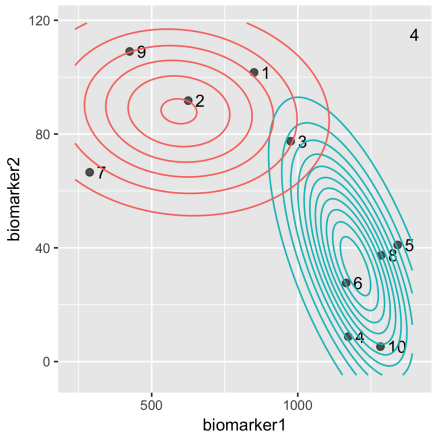
$$\mu_A = \begin{bmatrix} 1025.3 \\ 44.2 \end{bmatrix}$$

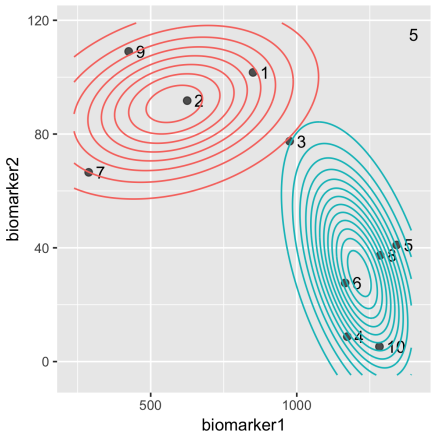
$$\mu_B = \begin{bmatrix} 672.9 \\ 87 \end{bmatrix}$$

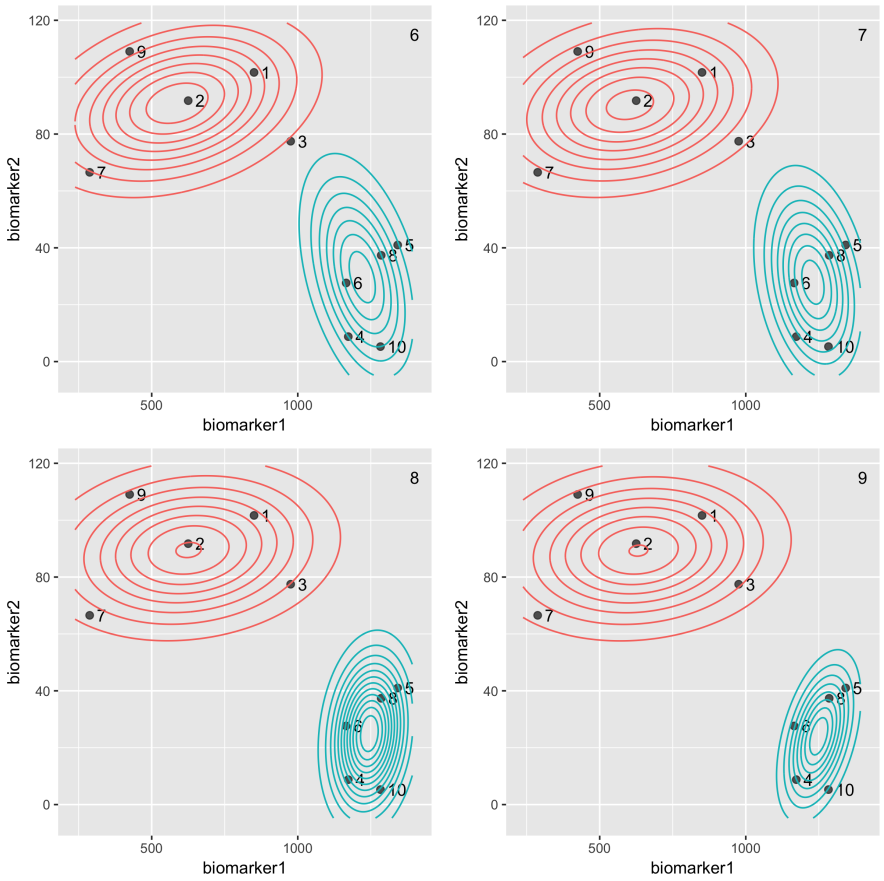
$$\mathbf{\Sigma}_A = \begin{bmatrix} 235.5^2 & -0.916 \cdot 235.5 \cdot 30.3 \\ -0.916 \cdot 235.5 \cdot 30.3 & 30.3^2 \end{bmatrix}$$

$$\mathbf{\Sigma}_B = \begin{bmatrix} 110.6^2 & -0.558 \cdot 110.6 \cdot 14.6 \\ -0.558 \cdot 110.6 \cdot 14.6 & 14.6^2 \end{bmatrix}$$

8. Do EM rounds 4 through 9.







Here is the table of calculations for the E-step after round 9:

i	$x_1^{(i)}$	$x_2^{(i)}$	$\mathcal{N}(x^{(i)}\|\mu_A, \mathbf{\Sigma}_A)$	$\mathcal{N}(x^{(i)}\|\mu_B, \mathbf{\Sigma}_B)$	$w_A^{(i)}$	$w_B^{(i)}$
1	634.83	110.55	1.8e-40	2.6e-05	0.000	1.000
2	650.06	74.22	2.5e-28	7.1e-05	0.000	1.000
3	788.24	81.52	1.6e-21	5.8e-05	0.000	1.000
4	771.47	84.98	2.5e-23	7.7e-05	0.000	1.000
5	515.81	91.08	1.7e-43	3.4e-05	0.000	1.000
6	1101.23	31.05	0.00011	6e-13	1.000	0.000
7	649.32	77.05	5e-29	9.5e-05	0.000	1.000
8	652.89	97.16	2.2e-34	0.00012	0.000	1.000
9	1183.02	11.73	0.00011	6e-18	1.000	0.000
10	1238.45	33.46	0.00011	3.7e-16	1.000	0.000
sum					3.000	7.000

The values of the final parameters are:

$$
\begin{aligned}
\phi_A &= 0.30 \\
\phi_B &= 0.70 \\
\mu_A &= \begin{bmatrix} 1174.2 \\ 25.4 \end{bmatrix} \\
\mu_B &= \begin{bmatrix} 666.1 \\ 88.1 \end{bmatrix} \\
\mathbf{\Sigma}_A &= \begin{bmatrix} 56.4^2 & -0.009 \cdot 56.4 \cdot 9.7 \\ -0.009 \cdot 56.4 \cdot 9.7 & 9.7^2 \end{bmatrix} \\
&= \begin{bmatrix} 3176.8 & -5.0 \\ -5.0 & 94.6 \end{bmatrix} \\
\mathbf{\Sigma}_B &= \begin{bmatrix} 84.8^2 & -0.287 \cdot 84.8 \cdot 11.7 \\ -0.287 \cdot 84.8 \cdot 11.7 & 11.7^2 \end{bmatrix} \\
&= \begin{bmatrix} 7185.8 & -284.8 \\ -284.8 & 137.5 \end{bmatrix}
\end{aligned}
$$

Question 20.7

Compare these parameters to the values from the code that generated the data (Question 20.6). What do you notice?

Question 20.8

Think of 2-3 different unsupervised learning problems from biology or medicine where a mixture model makes sense, conceptually at least, for modeling the data. How would you set up the mixture model in each case?

20.7 The Expectation-Maximization Algorithm

Given a joint distribution $p(x, z|\theta)$ over observed variables X and latent variables Z, governed by parameters θ, the goal of the EM algorithm is to maximize the likelihood function $p(x|\theta)$ with respect to θ. Mixture models are an

example of the EM algorithm. Here is its general form:

1. Choose an initial setting for the parameters θ.
2. **E step.** For each i, set
$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)}, \theta)$$
3. **M step.** Set
$$\theta := \arg\max_{\theta} \sum_{i=1}^{n} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}|\theta)}{Q_i(z^{(i)})}$$
4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, return to step 2.

Why does this work? See Andrew Ng's lecture notes on the EM algorithm from CS229 at Stanford – they contain the clearest explanation I've seen and use the same notation as us.

20.8 Extensions

There are many different clustering algorithms. You can find a good summary of all the different ones here: https://en.wikipedia.org/wiki/Cluster_analysis. Some topics you might choose to investigate independently (or that we might look at together in the future) include:

- Hierarchical clustering (e.g. phylogenetic trees)
- Methods for choosing K (the number of clusters)
- Bayesian methods that introduce priors on the parameters in mixture models
- Biclustering

There are also many different types of mixture models that all use the EM algorithm for maximum likelihood optimization. You may want to check out the following (and many more):

- Hidden Markov Models (Baum-Welch algorithm)
- Inside-outside algorithm for induction of probabilistic context-free grammars
- Leicht and Newman's 2007 PNAS paper on finding network clusters using mixture models
- Alignment algorithms for genetic sequence data using HMMs

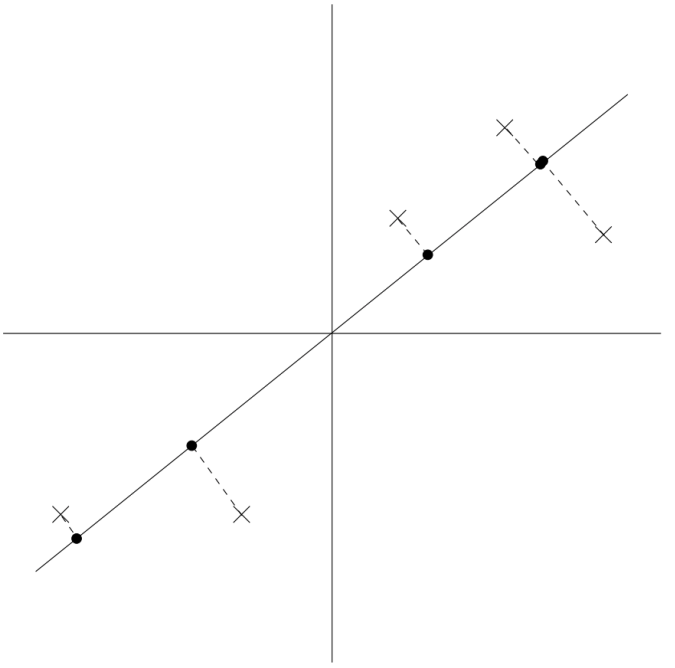
Chapter 21

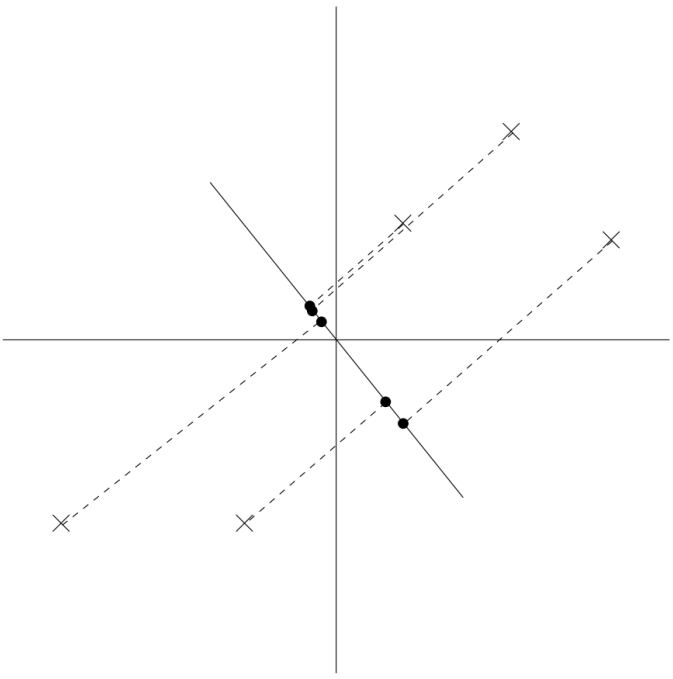
Principal Components Analysis

Clustering (Chapter 20) is one approach to uncovering latent structure in data. Another are matrix decomposition methods, the most famous of which is **principal components analysis (PCA)**. PCA is a powerful statistical tool for analyzing and visualizing datasets. It has been independently discovered several times over the course of history, and can be formulated mathematically a few different ways.

21.1 Principal Components: What are they?

You can think of PCA as a rotation of the feature space onto a set of axes that most efficiently represent the data. The **principal components** are these axes, or **basis vectors**. Here are pictures of the first two principal components for a small dataset. These were borrowed from Andrew Ng's lecture notes for CS229 at Stanford University.





Question 21.1

What do you notice about principal components 1 (top) and 2 (bottom)? Think in terms of (a) the variance (spread) of the data, and (b) the projection errors (dotted lines).

Mathematically, PCA is a linear projection of p-dimensional datapoints, $x^{(1)}, \ldots, x^{(n)}$ onto a k-dimensional space ($k \leq p$) defined by basis vectors $u_1, \ldots, u_k$ such that:

- the projection maximizes the variance from the original dataset that is retained
- the projection minimizes projection error (square loss)
- the basis vectors, $\{u_1, \ldots, u_k\}$, are orthogonal (i.e., perpendicular)

The number of distinct principal components is the smaller of the number of original variables (p) or the number of observations (n) minus one.

21.2 Definitions

21.2.1 Feature Vectors

Say you have n samples in a dataset, and each has dimensionality p. Let $x^{(i)}$ be the vector of p features for the ith person. We write

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_p^{(i)} \end{bmatrix}$$

and we can write our entire dataset as a $n \times p$ matrix like this:

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_p^{(n)} \end{bmatrix}$$

such that the ith row of the matrix is the vector of features for the ith person. This is exactly the same notation that we have used in previous chapters.

21.2.2 Variance and Covariance

The **sample covariance** of two features x_j and x_k is given by

$$\text{cov}(x_j, x_k) = S_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} \left(x_j^{(i)} - \overline{x_j} \right) \left(x_k^{(i)} - \overline{x_k} \right)$$

where the bar ($\overline{x_j}$) refers to the mean of x_j across the entire dataset. We can also write it in matrix format as

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^T.$$

The covariance can be any real number between positive and negative infinity. One weird thing about it is that it has units - the product of the units of the two features. This means that the covariance depends on the scale that is chosen for each feature, which is somewhat arbitrary. The **sample variance** of a given feature is just the covariance in cases where $j = k$ (the covariance of a feature with itself).

The **correlation** (technically the **Pearson correlation**) is a unitless, normalized version of the covariance and has the form

$$\text{cor}(x_j, x_k) = \frac{\sum_{i=1}^{n} (x_j^{(i)} - \overline{x_j})(x_k^{(i)} - \overline{x_k})}{\sqrt{\sum_{i=1}^{n} (x_j^{(i)} - \overline{x_j})^2} \sqrt{\sum_{i=1}^{n} (x_k^{(i)} - \overline{x_k})^2}}.$$

21.2.3 Orthogonality

Two vectors are **orthogonal** if their **inner product**, or **dot product**, is zero. This is defined as

$$\langle x, y \rangle = x \cdot y = \sum_{i=1}^{n} x_i y_i$$

where the sum is over the vector components. If this mathematical notation makes you uncomfortable, just think of "orthogonality" as "perpendicularity" and you'll be able to understand it visually.

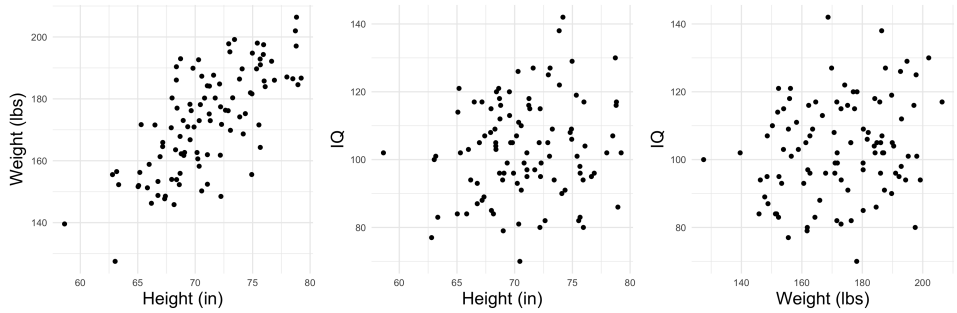
In PCA, the vectors whose inner products we are concerned with will *not* be the feature vectors for individual samples. They will be, instead, the *columns* of X – the values of a single feature for all n samples.

21.3 Example: Running PCA in R

Let's say you have measurements of weight, height, and IQ ($p = 3$) for $n = 100$ men. Here are the first 10 rows of the 100 row dataset:

Patient ID (i)	Height (in)	Weight (lbs)	IQ
1	74.9	155.6	109
2	66.5	171.6	117
3	74.4	175.2	91
4	65.8	151.3	84
5	70.2	162.7	107
6	78.8	206.4	117
7	69.1	162.7	96
8	70.6	150.3	110
9	65.2	156.2	121
10	66.8	153.3	93

Here are some scatterplots showing pairwise comparisons of the three features:



The **correlation matrix** for these data looks like this:

```
       height weight     iq
height  1.000  0.790 -0.103
weight  0.790  1.000 -0.078
iq     -0.103 -0.078  1.000
```

Question 21.2

Looking at the correlation matrix, which features are most tightly correlated? What does this imply about the direction of the first principal component, PC1?

21.3.1 Centering and Scaling

PCA is sensitive to the relative scales of the different variables in the original dataset. For this reason, datasets are usually *scaled* and *centered* before PCA is performed. All this means is that for each column (feature) of the dataset, we subtract its mean and divide by its standard deviation. The transformed column will have mean 0 and standard deviation 1.

If the data are scaled and centered, we can rewrite S (the sample covariance matrix from Section 21.2.2) as $X^T X/(n-1)$.

Question 21.3

What would happen to the principal components if you didn't center and scale the data?

Question 21.4

Why do you think the interpretation of the principal components becomes more difficult if you have features measured on lots of different scales (e.g., some categorical, some numeric/roughly normal, some numeric/highly skewed)?

21.3.2 Input and Output

In R, we can run PCA on our dataset, `d`, using the following command:

```
p <- prcomp(d[,2:4], center = TRUE, scale. = TRUE, rank. = 3)
```

where `rank.` is an optional parameter indicating the number, *k*, of principal components desired. The output looks like this:

```
> p$sdev
[1] 1.3454074 0.9899765 0.4580672

> p$rotation
              PC1         PC2         PC3
height  0.6996236 -0.09446119 -0.70823998
weight  0.6971414 -0.12698944  0.70559726
iq     -0.1565906 -0.98739595 -0.02299201

> p$center
   height    weight        iq
 70.74468 174.43346 101.12000

> p$scale
   height    weight        iq
 3.914989 13.853272 14.183701
```

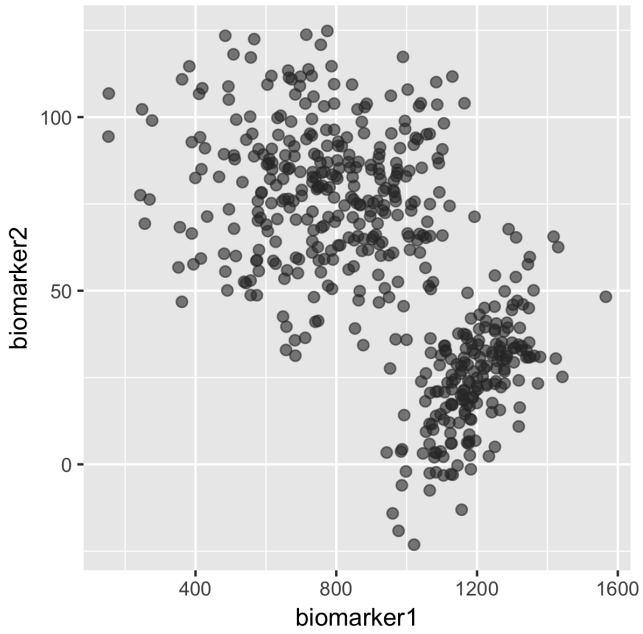
```
> p$x
              PC1         PC2         PC3
 [1,] -0.189268854  1.976180297 -0.416704298
 [2,]  1.794925031  0.710160037  0.575478783
 [3,] -0.404617344  0.283418147  0.151536071
 [4,] -0.471675025 -0.710146894  0.347006397
 [5,] -1.073877096 -0.234684815 -0.638243675
 [6,] -0.654663317  0.043481120 -0.107082277
 [7,] -1.705841441  0.368990046 -0.794047654
 [8,]  2.311353981 -0.501565781  0.033112309
 [9,] -0.006387264 -0.535422493 -1.126177513
[10,] -1.557136911  1.399023167 -0.045281059
[11,]  2.632974868 -1.046037442 -0.239934797
[12,] -0.978337630 -0.193476727  0.007814938
[13,] -0.625980903 -0.456274882 -0.305349616
(continued)
```

Question 21.5

Describe/draw the directions of the three principal component vectors, PC1, PC2, and PC3, in the coordinate system of the original predictors, height, weight, and IQ.

Question 21.6

Here is a picture of the flow cytometry dataset we first encountered in Chapter 20.



What would PC1 and PC2 look like for this dataset? (Why is there no PC3?) How could PCA help you separate the two clusters?

21.4 Applications of PCA

21.4.1 Eigenfaces

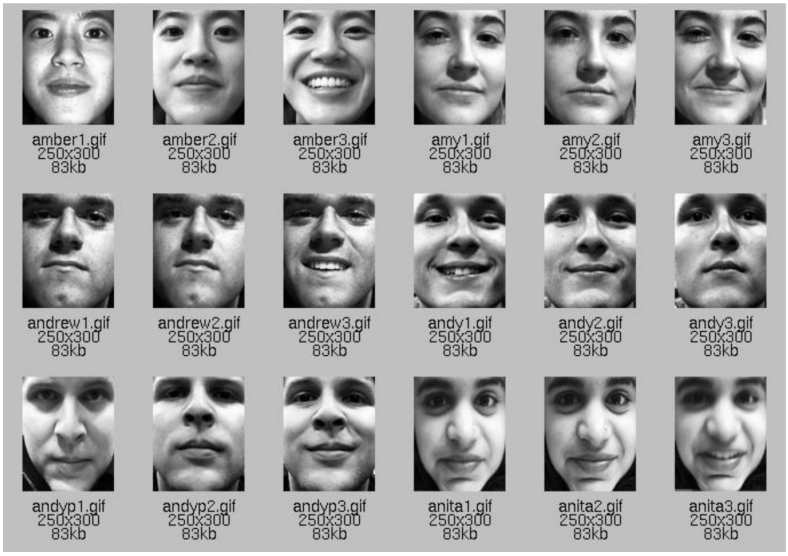
Eigenfaces were an early application of PCA to computer vision, specifically image search and retrieval. Here's what you do to create a set of eigenfaces:

1. Prepare training set of images taken under the same lighting conditions, with mouths and eyes aligned, resampled to a common pixel resolution.
2. Generate a vector for each image by concatenating the pixel intensities

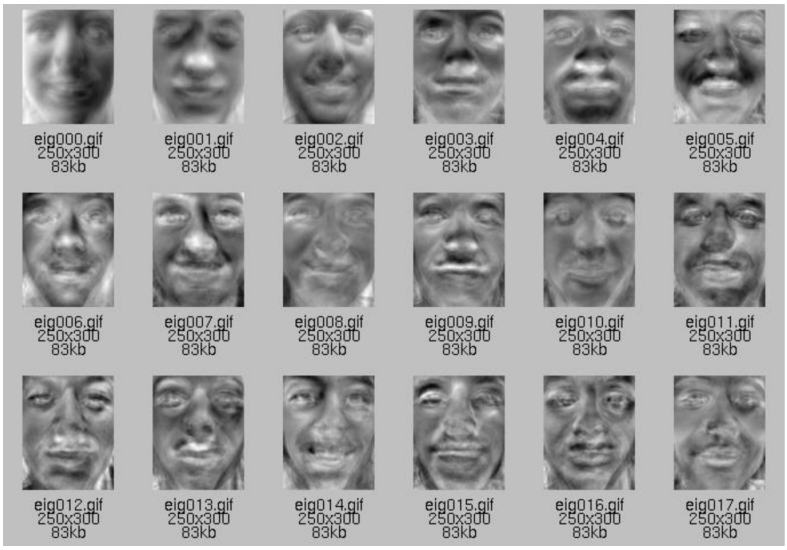
across the rows. So if the image is of dimension $r \times c$, the image vector will have $p = rc$ features.

3. Create the data matrix from the n vectors.
4. Center the data (Why wouldn't we scale the data?).
5. Calculate the eigenvectors and eigenvalues of the sample covariance matrix. Each eigenvector corresponds to one eigenface.

Some real eigenfaces are shown below [1].



[1] Figure details: (top) some of the original faces from the training set (there were 86 images total); (bottom) the eigenfaces corresponding to the 18 largest eigenvalues of the covariance matrix. From https://www.clear.rice.edu/elec301/Projects99/faces/images.html.



Question 21.7

What is X for the eigenfaces problem? What are the principal components? How could you use the principal components to match a new face to an existing database of faces?

21.4.2 Image Compression

PCA can also be used for image compression. In that case, the raw pixels for the image are the data matrix, X (so each "sample" is one row of the image and each "feature" is one column of the image). Generally the data are centered but not scaled here (Why?). A subset of the principal components corresponding to the r largest eigenvalues are used to reconstruct the image. The picture below illustrates the process[2].

[2]Figure: Using PCA for image compression. The image is recomposed by adding together many gridlike images like that shown in (a). From `https://www.projectrhea.org/rhea/index.php/PCA_Theory_Examples`.

Question 21.8

What is X for the image compression problem? What are the principal components? How does using PCA help compress the image?

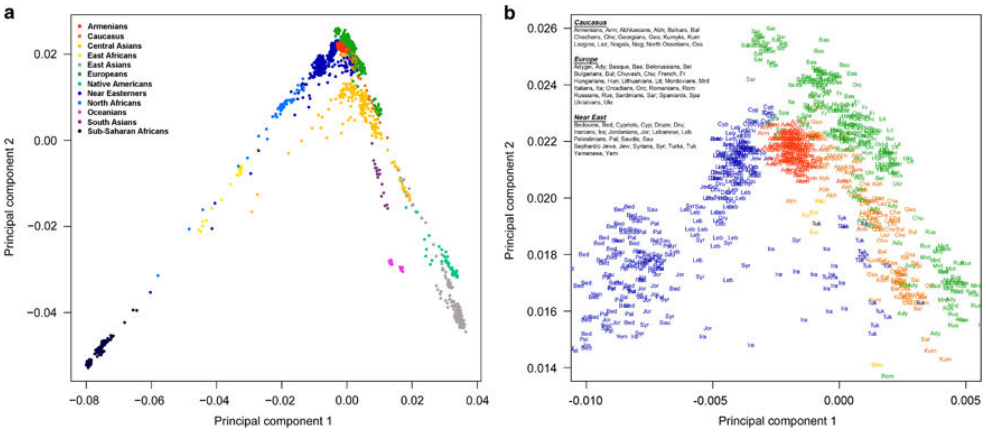
21.4.3 Genetic Ancestry

There are many awesome population genetics papers that use PCA to locate individuals within the "space" of possible genetic mutations. Here's some text from one, a figure from which is shown below[3]. Here's a quote from the original paper:

> Here, we analyse genome-wide variation in 173 Armenians and compare them with 78 other worldwide populations. We find that Armenians form a distinctive cluster linking the Near East,

[3]From European Journal of Human Genetics (2016) 24, 931-936 (2016).

Europe, and the Caucasus. We show that Armenian diversity can be explained by several mixtures of Eurasian populations that occurred between 3000 and 2000 BCE, a period characterized by major population migrations after the domestication of the horse, appearance of chariots, and the rise of advanced civilizations in the Near East. However, genetic signals of population mixture cease after 1200 BCE when Bronze Age civilizations in the Eastern Mediterranean world suddenly and violently collapsed. Armenians have since remained isolated and genetic structure within the population developed 500 years ago when Armenia was divided between the Ottomans and the Safavid Empire in Iran.



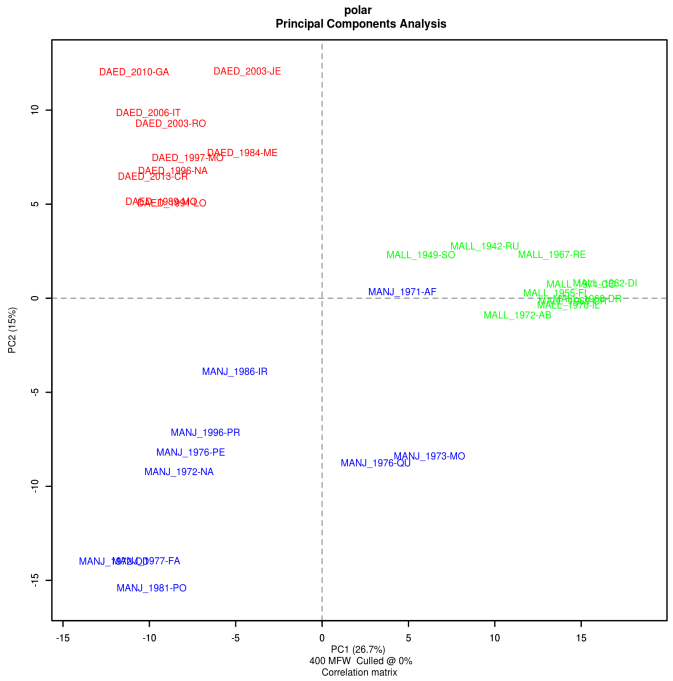
Question 21.9

What is X for the genetic ancestry problem? What are the principal components? How were the principal components used to produce the figure above?

21.4.4 Topic Modeling

PCA also has many uses in natural language processing. It forms the basis for latent semantic indexing (see Deerwester 1990) and can also be used for topic modeling, a form of mixture model (see Section 20.3)[4].

[4]Figure: Topic modeling French crime fiction. From https://dragonfly.hypotheses.org/530.



Question 21.10

What is X for the topic modeling problem? For the latent semantic indexing problem? What do the principal components represent?

Question 21.11

Think of 2-3 different unsupervised learning problems from biology or medicine where PCA makes sense, conceptually at least, for modeling the data. How would you set up the data matrix in each case? What would the principal components correspond to in the data?

21.5 Technical Details (Advanced)

21.5.1 Eigenvalues and Eigenvectors

One interpretation of matrix multiplication, Ax, where A is an $m \times m$ square matrix and x a vector of length m, is that A linearly transforms x by rotating it,

changing its magnitude, or both. An **eigenvector** of A is a vector that, when multiplied by A, grows or shrinks in magnitude but does not rotate. The multiplier of its magnitude is given by the corresponding **eigenvalue**. There are m different eigenvalues and eigenvectors for an $m \times m$ matrix, although the eigenvalues may not all be distinct. The set of all eigenvalues of A is called the **spectrum** of A.

For all eigenvalue-eigenvector pairs, the relationship $Av = \lambda v$, where v is an eigenvector and λ its corresponding eigenvalue, must hold. To find the λs and vs analytically, we can set the determinant $|A - \lambda I|$, called the **characteristic polynomial**, equal to zero and solve for the eigenvalue and eigenvector corresponding to each root. Note that any scalar multiple of an eigenvector is itself an eigenvector; usually we restrict eigenvectors to have magnitude 1 for this reason.

If a matrix is **symmetric** and **positive semidefinite** (all positive or zero eigenvalues), all of its eigenvectors will be orthogonal.

21.5.2 PCA: Eigendecomposition Version

The first way we can find the principal components is by performing an eigendecomposition identical to the one we performed in Section 21.5.1 on the sample covariance matrix. The principal components will be the eigenvectors of this matrix, and the corresponding eigenvalues tell you how much of the dataset's overall variance is accounted for by each eigenvector.

One useful fact about the covariance matrix is that, because it is a symmetric matrix, it can be diagonalized:

$$S = \frac{1}{n-1} X^T X = V L V^T \tag{21.1}$$

where V is a matrix of the p eigenvectors of S (each column is an eigenvector) and L is a diagonal matrix with the eigenvalues of S along the diagonal. This fact will be important later.

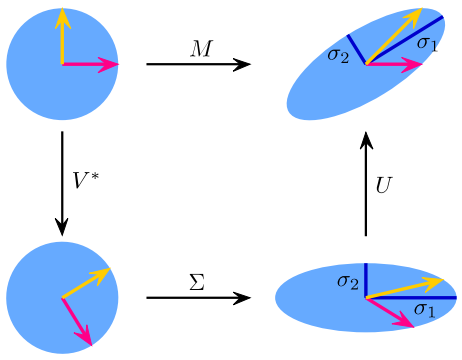
21.5.3 PCA: Singular Value Decomposition (SVD) Version

Although the eigendecomposition of the covariance matrix is generally the way PCA is presented, in practice most software uses another matrix decomposition called the **singular value decomposition (SVD)**, even though it is slower to compute, because it's more numerically stable.

Here's why the two methods are equivalent. Any matrix, M, with real or complex values (we'll focus on real valued matrices here) has an SVD of the form[5]:

$$M = UDV^T$$

where U is an orthogonal matrix, meaning that $U^TU = I$, D is diagonal and all of its nonzero elements are non-negative real numbers called **singular values**, and V is also orthogonal. For a square matrix, the SVD can be viewed as "breaking up" the transformation described by a matrix into three separate transformations: a rotation/reflection, a scaling, and another rotation/reflection[6]:



$$M = U \cdot \Sigma \cdot V^*$$

[5]This is in contrast with the eigendecomposition, which works only on positive semidefinite matrices.

[6]Figure: Graphical representation of the SVD of a square matrix, M. Note that the illustrator here uses Σ instead of D for the middle, diagonal matrix. Attribution: By Georg-Johann (Own work), via Wikimedia Commons.

If we perform the SVD on X, we obtain a decomposition

$$X = UDV^T$$

where D is a diagonal matrix with singular values $d_1, \ldots, d_p$. Knowing this, we can rewrite our covariance matrix as

$$\begin{aligned} S &= \frac{1}{n-1} X^T X \\ &= \frac{1}{n-1} (UDV^T)^T UDV^T \\ &= \frac{1}{n-1} VDU^T UDV^T \\ &= V \frac{D^2}{n-1} V^T. \end{aligned}$$

Comparing this to the form from Equation 21.1, we see that the eigenvectors correspond to the right singular values of the SVD, and the eigenvalues are related to the singular values on the diagonal of D by $\lambda_i = d_i^2/(n-1)$.

There are a lot of interpretations of the SVD. The one I like best is that the SVD provides a "nearest orthogonal matrix" to the original matrix. It's like the data have a coordinate system of orthogonal axes in which they most like to live, and the SVD tells you what that system is.