Unsupervised Learning

Justin Post

You might recall our very early discussion of the difference between supervised and unsupervised learning: unsupervised learning does not include a response variable.

This makes unsupervised learning an inherently more difficult task than supervised learning. We don’t have a value to compare against. We are building a model to describe patterns and relationships in the data, but we can’t really verify the structure!

Now we’ll look at two common unsupervised techniques in more detail:

* Clustering
  + K Means
  + Heirarchical
* Principal Components Analysis (PCA)

## Clustering

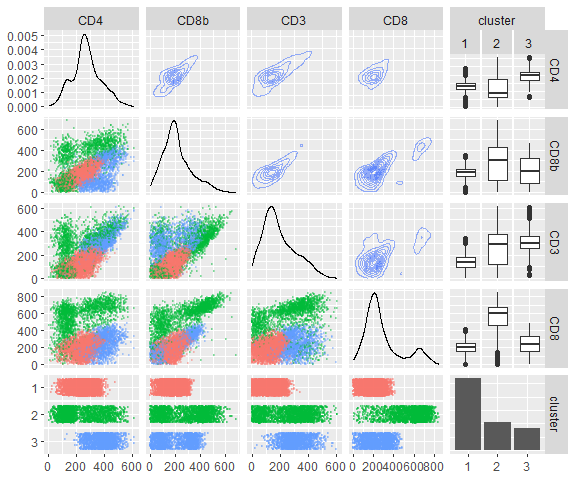
We looked at an example of clustering on a data set about Graft-versus-Host-Disease (GvHD) from the mclust package in R (Srucca, et al., 2023). This data set has:

Two samples of this flow cytometry data, one from a patient with the GvHD, and the other from a control patient. The GvHD positive and control samples consist of 9083 and 6809 observations, respectively. Both samples include four biomarker variables, namely, CD4, CD8b, CD3, and CD8. The objective of the analysis is to identify CD3+ CD4+ CD8b+ cell sub-populations present in the GvHD positive sample.

We attempted to group similar observations together via a clustering algorithm and then tried to interpret the grouping.

library(mclust)  
clusters <- Mclust(GvHD.pos, G = 3)  
GvHD\_cluster <- GvHD.pos |>  
 mutate(cluster = as.factor(clusters$classification))  
GvHD\_cluster[1:5, ] |>  
 knitr::kable()

| CD4 | CD8b | CD3 | CD8 | cluster |
| --- | --- | --- | --- | --- |
| 308 | 220 | 157 | 339 | 1 |
| 319 | 271 | 223 | 350 | 1 |
| 215 | 148 | 119 | 221 | 1 |
| 104 | 49 | 284 | 178 | 2 |
| 262 | 167 | 144 | 156 | 1 |



Exploratory graphs of the four variables under consideration with cluster groupings included. A four by four matrix of graphs is shown similar to the previous figure. However, this figure includes a coloring according to the cluster associated with each observation (1, 2, or 3). With this, the groupings of points noted earlier are now clearly placed into different clusters indicating an ability for a subject matter expert to interpret the results.

Again, we note that without a response variable we do not have a great way to *check* results of our model.

* There is no correct answer to look at with test data
  + Groups should have members that are “similar” to one another
  + # of subgroups subjective most of the time
* We hope that patterns detected extend to future data but checking that is difficult
* Overall, unsupervised learning is pretty subjective and generally considered exploratory

Let’s look at how we form the clusters in more detail.

* Two major clustering methods (with a lot of variants)
  + K-Means clustering
  + Hierarchical clustering

### K Means Clustering

As we want the observations in our clusters (or groups) to be as similar as possible, we can investigate the *within-cluster variation* as a way to determine cluster membership.

The idea is as follows:

* Place all observations in one of clusters
* For kth cluster, sum all pairwise squared Euclidean distances between the observations in the kth cluster. Divide by # of observations.
* Essentially, this gives us the average of distances between all pairs of points!
* We find this quantity for each cluster, sum them all together, and try to minimize this quantity

As you might guess, it is a reasonably difficult problem to find the ‘optimal’ clusters. The most commonly used algorithm used finds a *local minimum* for the function. The algorithm is as follows:

1. Set the number of clusters to .
2. Randomly assign a number, from 1 to K, to each of the observations. (These serve as initial cluster assignments for the observations.)
3. Iterate the following until the cluster assignments stop changing:
   * For each of the K clusters, compute the mean for each variable across the values in that cluster (called a centroid).
   * Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance or using some other metric).

Since we are not likely to find a global min, we are (probably) finding a local min when we run this algorithm. That means if we run the algorithm again we may not get the same clusters!

Usually we run the algorithm many times and take the run with the overall smallest objective function (sum of average cluster distances) as our final clustering assignment.

#### Visual of K Means Clustering

Let’s just play around with the simple iris data set that contains information about 150 flowers. Here we have a Species of the flower that we will ignore (this could represent an actual grouping that we might be trying to find in the underlying data).

iris[1:5, ] |>  
 knitr::kable()

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |

my\_iris <- iris |>  
 select(-Species)

Now we can use the kmeans() function from the mclust package. We’ll use just the Sepal measurements, strictly so we can plot the results.

As this uses a random starting point for the clusters, we’ll set a seed to get reproducible results.

set.seed(10)  
iris\_clusters <- kmeans(my\_iris[, 1:2], centers = 3, algorithm = "MacQueen")

Here centers = 3 specifies that we want to split the data into three groups or clusters.

We can see that each observation is assigned to a cluster:

my\_iris <- my\_iris |>   
 mutate(cluster = iris\_clusters$cluster)  
my\_iris[1:5, c(1:2, 5)] |>  
 knitr::kable()

| Sepal.Length | Sepal.Width | cluster |
| --- | --- | --- |
| 5.1 | 3.5 | 1 |
| 4.9 | 3.0 | 1 |
| 4.7 | 3.2 | 1 |
| 4.6 | 3.1 | 1 |
| 5.0 | 3.6 | 1 |

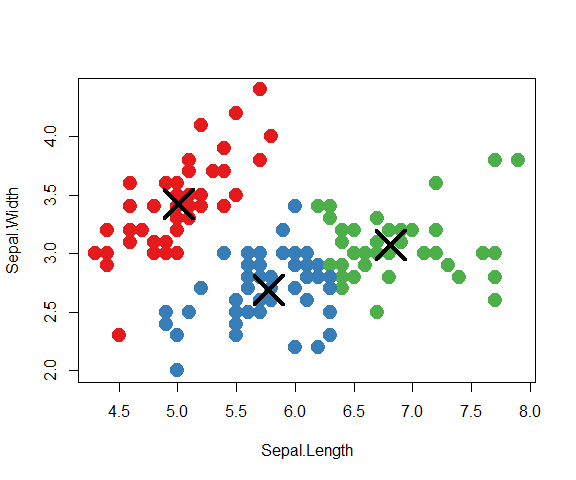
The **centroids** are given by

iris\_clusters$centers |>  
 knitr::kable()

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.006000 | 3.428000 |
| 5.773585 | 2.692453 |
| 6.812766 | 3.074468 |

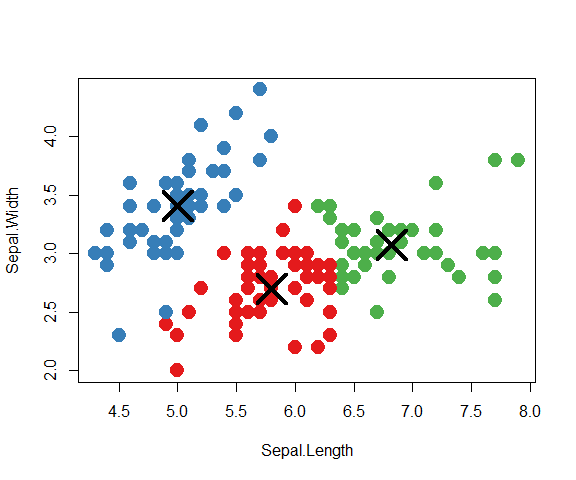
Let’s plot these cluster assignments!

palette(c("#E41A1C", "#377EB8", "#4DAF4A"))  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3)  
points(iris\_clusters$centers, pch = 4, cex = 4, lwd = 4)



Note, if we run this again we may get slightly different clusters!

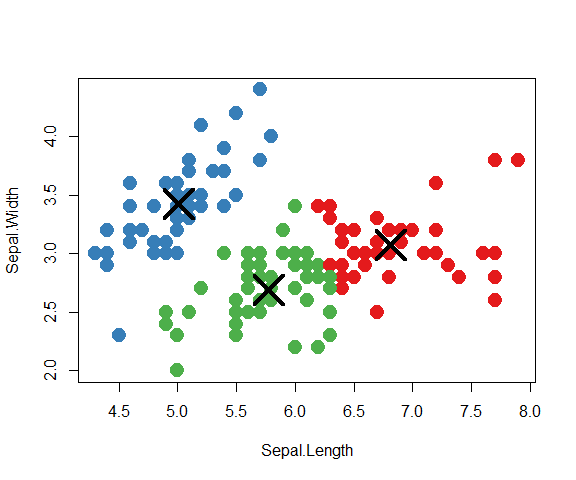
set.seed(511)  
iris\_clusters <- kmeans(my\_iris[, 1:2], centers = 3, algorithm = "MacQueen")  
my\_iris <- my\_iris |>   
 mutate(cluster = iris\_clusters$cluster)  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3)  
points(iris\_clusters$centers, pch = 4, cex = 4, lwd = 4)



The point at (4.8, 2.5) or so differs in terms of its cluster assignment!

We can run the algorithm with multiple starting values by modifying the nstart argument. We then should get more stable results.

set.seed(511)  
iris\_clusters <- kmeans(my\_iris[, 1:2],   
 centers = 3,   
 algorithm = "MacQueen",  
 nstart = 11,   
 iter.max = 30)  
my\_iris <- my\_iris |>   
 mutate(cluster = iris\_clusters$cluster)  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3)  
points(iris\_clusters$centers, pch = 4, cex = 4, lwd = 4)



### Hierarchical Clustering

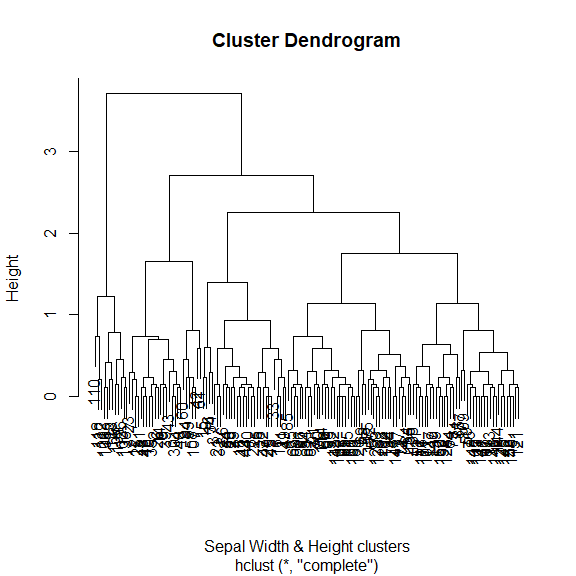
One potential drawback of kmeans clustering is that a structure across the number of clusters is not imposed. We’ll explain this idea through the alternative clustering method called hierarchical clustering.

In this type of clustering we use a *bottom up* approach. Here we do not need to specify a # of clusters. The idea is as follows:

* Start with all observations in their own cluster
* Join the ‘closest’ observations (lessening the number of clusters each time), until we get down to 1 cluster.
* We can visualize the clusters with a dendrogram

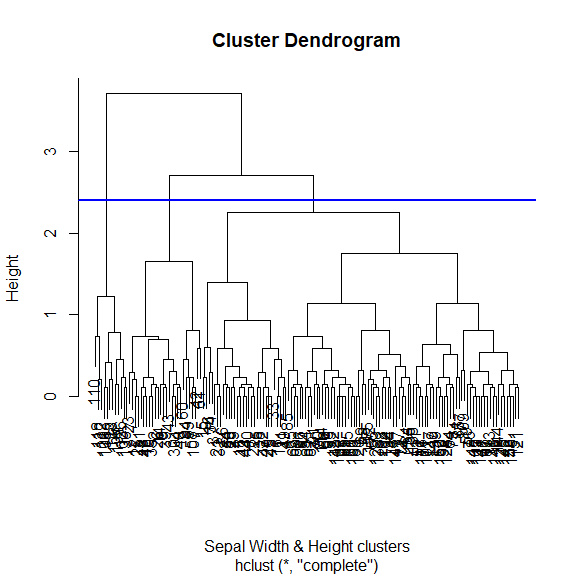
Let’s look at an example to get us started. Here we use the hclust() function from the stats library.

iris\_hier\_clust <- hclust(dist(data.frame(iris$Sepal.Length, iris$Sepal.Width)))  
plot(iris\_hier\_clust, xlab = "Sepal Width & Height clusters")



We can see at the bottom of the dendrogram that each observation is in its own cluster.

We can determine cluster membership Using ‘Horizontal Line’



As we move up the dendrogram, we have fewer clusters and observations grouped together. That is, clusters are nested in some sense (hence the term hierarchical).

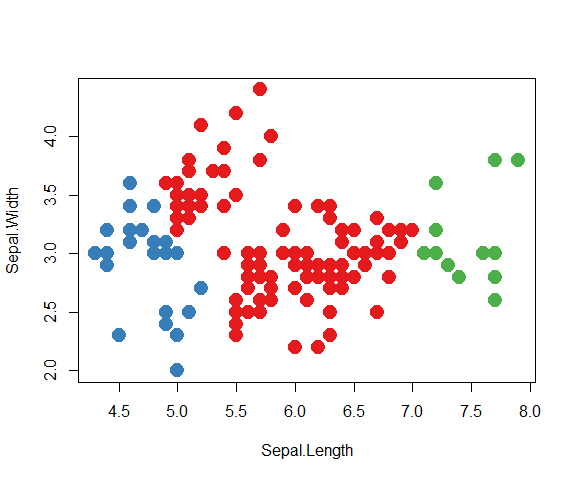
This implies that the five group cluster can be rolled up into the four group cluster. This is not the case in kmeans clustering!

We can pull out a given cluster assignment

head(cutree(iris\_hier\_clust, 3))

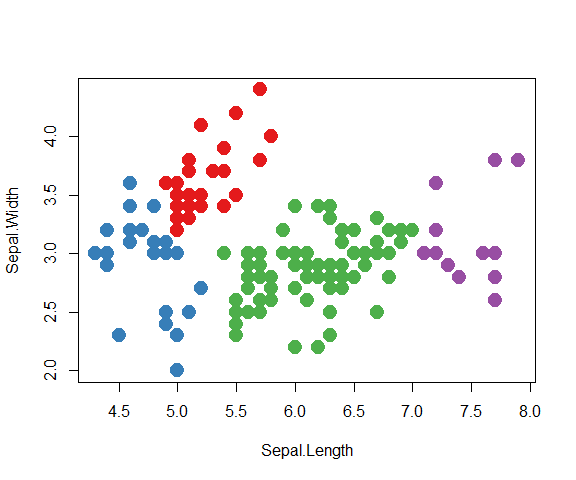
[1] 1 2 2 2 1 1

my\_iris <- my\_iris |>   
 mutate(cluster = cutree(iris\_hier\_clust, 3))  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3)



The four group cluster splits one of these clusters into two!

palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3"))  
my\_iris <- my\_iris |>   
 mutate(cluster = cutree(iris\_hier\_clust, 4))  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3)



#### Linkage in Agglomerative Clustering

When doing **agglomeration** different methods can be used to *join* clusters/groups. These methods are based on a **linkage**.

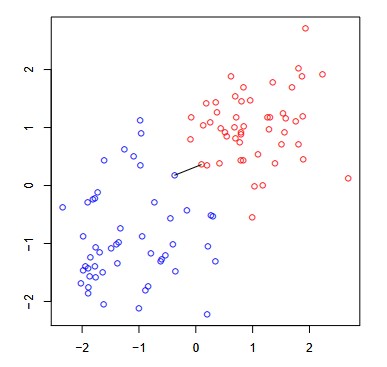
Linkage is a measure of dissimilarity between two groups. Define the linkage between two groups, and as . The algorithm for determing the clusters is:

1. Start with all points in their own groups
2. Until there is only one cluster, repeatedly: merge the two groups and such that is smallest

When plotting, the height on the dendrogram is proportional to the dissimilarity between nodes.

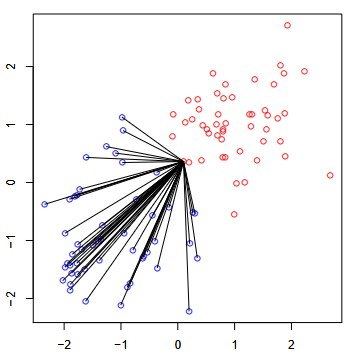
Some common linkage metrics are:

* Complete (default for hclust())
  + Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the largest of these dissimilarities.
* Single (nearest-neighbor linkage)
  + Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the smallest of these dissimilarities.
  + Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.



A scatterplot between two variables is shown with cluster membership given by color. The two groups are completely separated. The ‘single’ linkage score is given by the closest two observations from each group. This is shown by the connected line between the point from the blue cluster and the point between the red cluster. Taken from https://www.stat.cmu.edu/~ryantibs/datamining/lectures/05-clus2.pdf

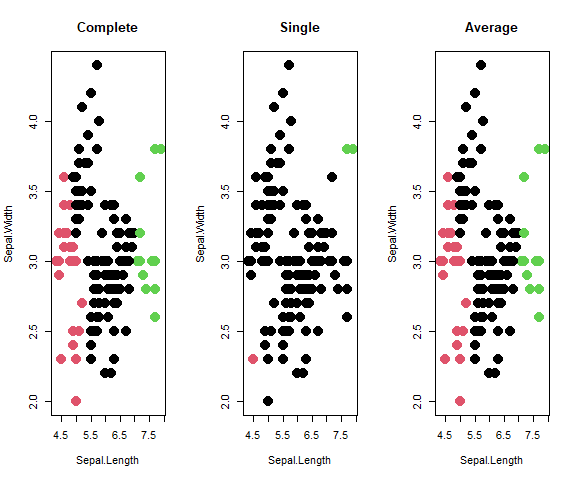
* Average
  + Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the average of these dissimilarities.



A scatterplot between two variables is shown with cluster membership given by color. The two groups are completely separated. The ‘average’ linkage score is given by average distance across all piars of points. The average distance for a single point from the red group is shown but all the lines from the blue group being connected to it. Taken from https://www.stat.cmu.edu/~ryantibs/datamining/lectures/05-clus2.pdf

Let’s implement these and see the clusters formed.

iris\_hier\_clust\_single <- hclust(dist(data.frame(iris$Sepal.Length,  
 iris$Sepal.Width)),  
 method = "single")  
  
iris\_hier\_clust\_average <- hclust(dist(data.frame(iris$Sepal.Length,  
 iris$Sepal.Width)),  
 method = "average")  
  
my\_iris <- my\_iris |>   
 mutate(cluster = cutree(iris\_hier\_clust, 3),  
 cluster\_single = cutree(iris\_hier\_clust\_single, 3),  
 cluster\_average = cutree(iris\_hier\_clust\_average, 3))  
par(mfrow = c(1, 3))  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3,   
 main = "Complete")  
plot(my\_iris[, 1:2],  
 col = my\_iris[,6],  
 pch = 20, cex = 3,  
 main = "Single")  
plot(my\_iris[, 1:2],  
 col = my\_iris[,5],  
 pch = 20, cex = 3,  
 main = "Average")



From <https://www.stat.cmu.edu/~ryantibs/datamining/lectures/05-clus2.pdf>

Single and complete linkage can have some practical problems:

* Single linkage suffers from *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 *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

Note: We could also do a *top-down* approach to hierarchical clustering (called *divisive*).

### Clustering Recap

Kmeans clustering requires us to specify the number of clusters and we iteratively try to find the optimal value.

* Can get different clusters in each run. Good to run the algorithm multiple times and take the best run
* Clusters are not nested

Hierarchical clustering can be done (bottom up is easiest). We start with each observation in its own cluster and join them one at a time.

* Specify a *dissimilarity* measure called the linkage
* Dendrogram provides a nice visualization

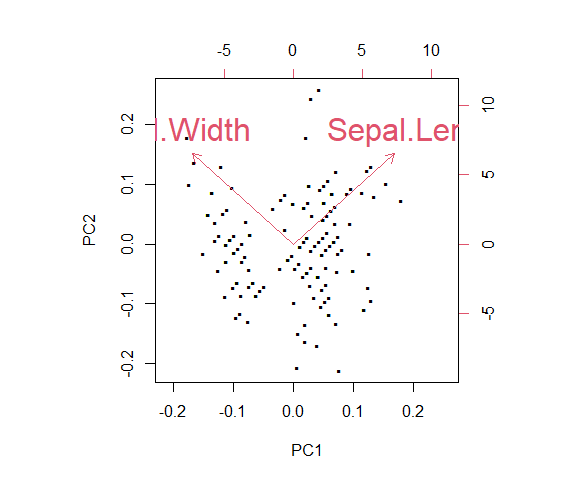
## Principal Components

Principal Components Analysis (PCA) is a dimension reduction technique. This tries to find patterns in the data in a very different way from clustering. Idea:

* If you have variables, they contain some joint variability/correlation
* PCA looks for *linear combinations* of those variables that account for most of the variability
* Each subsequent linear combination is independent of the previous linear combinations
* Usually can represent most of the variability of the variables by PCs
* Reduces # of variables from to
* Can be useful for visualizations
* Can use PCs as predictors in a regression model (PC Regression)

Let’s visualize the idea in two dimensions.

PCs <- prcomp(iris |> select(starts\_with("Sepal")),  
 scale = TRUE)  
biplot(PCs, xlabs = rep(".", nrow(iris)), cex = 2)



We can see the linear combinations found and the amount of variation as well.

PCs

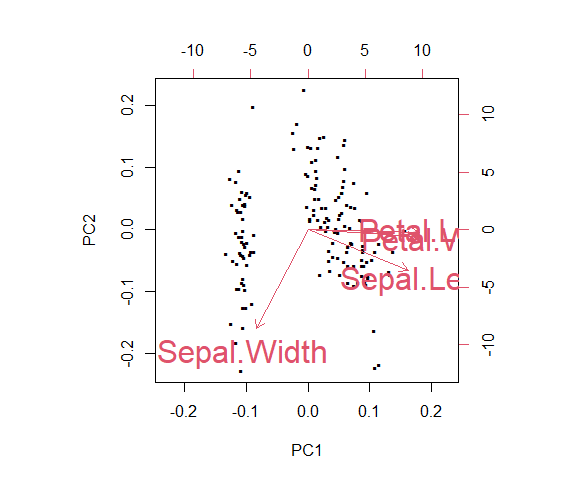
Standard deviations (1, .., p=2):  
[1] 1.0571517 0.9393776  
  
Rotation (n x k) = (2 x 2):  
 PC1 PC2  
Sepal.Length 0.7071068 0.7071068  
Sepal.Width -0.7071068 0.7071068

Of course this is more useful with more variables. Let’s consider all four numeric variables from the iris data set. How could we represent this in two dimensions?

PCs <- prcomp(iris[, -5], center = TRUE, scale = TRUE)  
PCs

Standard deviations (1, .., p=4):  
[1] 1.7083611 0.9560494 0.3830886 0.1439265  
  
Rotation (n x k) = (4 x 4):  
 PC1 PC2 PC3 PC4  
Sepal.Length 0.5210659 -0.37741762 0.7195664 0.2612863  
Sepal.Width -0.2693474 -0.92329566 -0.2443818 -0.1235096  
Petal.Length 0.5804131 -0.02449161 -0.1421264 -0.8014492  
Petal.Width 0.5648565 -0.06694199 -0.6342727 0.5235971

biplot(PCs, xlabs = rep(".", nrow(iris)), cex = 2)



The standard deviations associated with each PC gives the variability accounted for. The sum of the variances gives the total variation in the original data (4 here since we standardized the data).

sum(PCs$sdev^2)

[1] 4

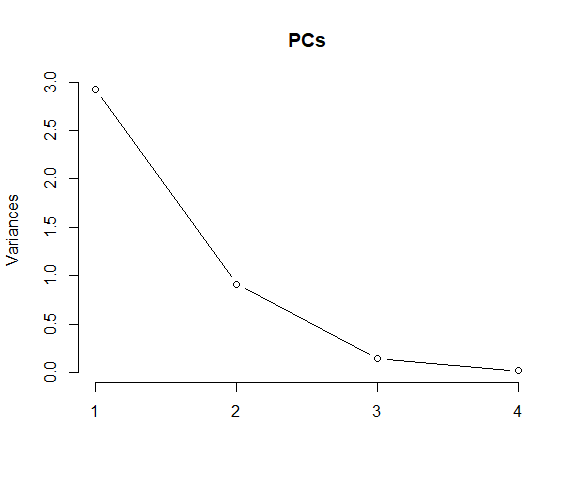
diag(cov(iris[,-5] |> scale())) |>   
 sum()

[1] 4

We often look at the proportion of variation given by the first PCs to obtain say, 80 or 90% of the total variation in the data. Then these variables represent the dimension reduced data!

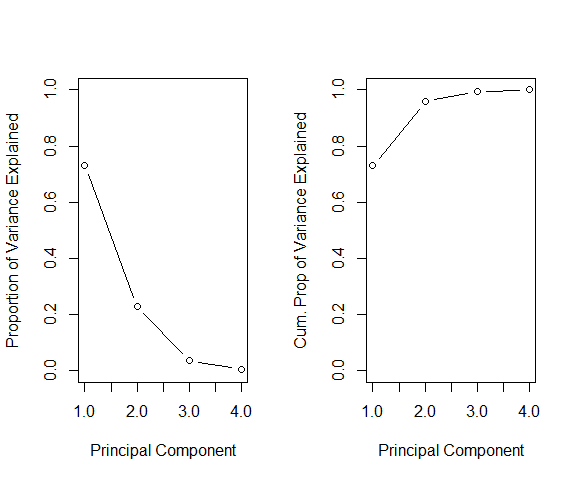
A *scree plot* is usually created and we look for an ‘elbow’.

screeplot(PCs, type = "lines") #scree plot used for visual



Alternatively, we can plot these ourselves and give the cumulative variation explained.

par(mfrow = c(1, 2))  
plot(PCs$sdev^2/sum(PCs$sdev^2),   
 xlab = "Principal Component",   
 ylab = "Proportion of Variance Explained",   
 ylim = c(0, 1),   
 type = 'b')  
plot(cumsum(PCs$sdev^2/sum(PCs$sdev^2)),   
 xlab = "Principal Component",  
 ylab = "Cum. Prop of Variance Explained",   
 ylim = c(0, 1),   
 type = 'b')



### PCA Details

Our goal with PCA is to obtain linear combinations of the variables that account for the largest amount of variability.

* After finding the first linear combination, we require that the second be orthogonal to the first.
* After finding the first two linear combinations, we require that the third be orthogonal to the first two.
* We continue until we obtain linear combinations.

#### Finding the first linear combination

Consider a linear combination of our variables:

* Here the $\phi\_$ are constants we need to determine
* We must constrain values, otherwise our variance could get arbitrarily large
* Our constraint

is called *normalizing* the components.

These terms are called the **loadings** and help us understand how much each predictor plays a role in the particular principle component.

As the variation doesn’t depend on the center, we can assume that each predictor has mean 0 (we generally standardize anyway!). This gives us

For a sample of size , we have

We try to maximize the sample variance of this linear combination (we divide by since the means are known to be 0), subject to the normalizing constraint:

Once first PC is found, we now find next most variable linear combination where this linear combination is uncorrelated with the first PC.

* Here, uncorrelated is equivalent to being perpendicular to the first. That is,

#### Relation to Eigendecomposition

Solving for the PCs is equivalent to doing an eigenvalue decomposition on the covariance or correlation (if we scale our predictors) matrix!

* Eigenvectors represent the “loadings”
  + Weight for each variable’s “importance” in the PC
  + Can be used to create interpretations of PCs if we’d like
* Eigenvalues represent how much of the variation exists on that PC
* Largest eigenvalue (with corresponding eigenvector) corresponds to first PC

The SDs given in output represent the square roots of the eigenvalues of the covariance/correlation matrix

### PCA Example

Take for example the banknote data in the mclust package. (See ?banknote for details.) The data contains six measurements made on 100 genuine and 100 counterfeit old-Swiss 1000-franc bank notes.

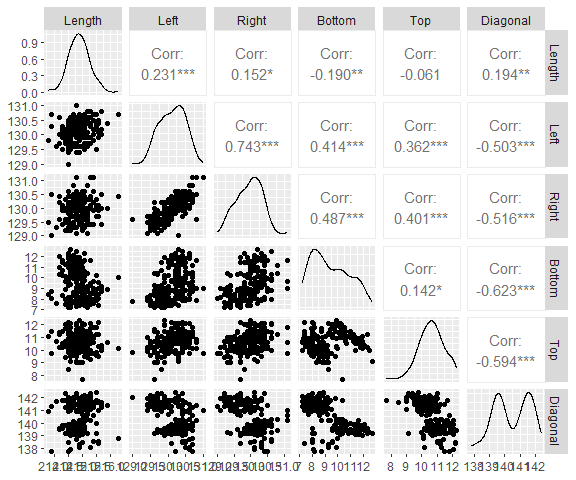
A snapshot of the data is shown below. (We have omitted the Staus variable which tells us whether a note is genuine or counterfeit. This is because unsupervised learning does not require a target/outcome variable.)

| Length | Left | Right | Bottom | Top | Diagonal |
| --- | --- | --- | --- | --- | --- |
| 214.8 | 131.0 | 131.1 | 9.0 | 9.7 | 141.0 |
| 214.6 | 129.7 | 129.7 | 8.1 | 9.5 | 141.7 |
| 214.8 | 129.7 | 129.7 | 8.7 | 9.6 | 142.2 |
| 214.8 | 129.7 | 129.6 | 7.5 | 10.4 | 142.0 |

A pairs plot of the data set is shown below.

GGally::ggpairs(swissTib)

Warning in geom\_point(): All aesthetics have length 1, but the data has 36 rows.  
i Please consider using `annotate()` or provide this layer with data containing  
 a single row.



After performing PCA, we can find two new features, engineered from the original six features, that capture roughly of the total variation (defined as the sum of variances of the variable present in the data) of the data.

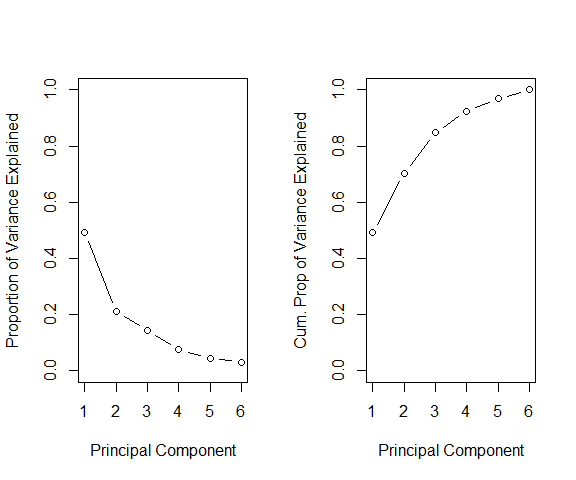
**As the scales of the variable’s are important when considering the variation, we usually standardize our data fully.** That is, we center and scale each predictor to have mean 0 and standard deviation 1.

PCs <- prcomp(swissTib, center = TRUE, scale = TRUE)  
PCs

Standard deviations (1, .., p=6):  
[1] 1.7162629 1.1305237 0.9322192 0.6706480 0.5183405 0.4346031  
  
Rotation (n x k) = (6 x 6):  
 PC1 PC2 PC3 PC4 PC5 PC6  
Length 0.006987029 -0.81549497 0.01768066 0.5746173 -0.0587961 0.03105698  
Left -0.467758161 -0.34196711 -0.10338286 -0.3949225 0.6394961 -0.29774768  
Right -0.486678705 -0.25245860 -0.12347472 -0.4302783 -0.6140972 0.34915294  
Bottom -0.406758327 0.26622878 -0.58353831 0.4036735 -0.2154756 -0.46235361  
Top -0.367891118 0.09148667 0.78757147 0.1102267 -0.2198494 -0.41896754  
Diagonal 0.493458317 -0.27394074 -0.11387536 -0.3919305 -0.3401601 -0.63179849

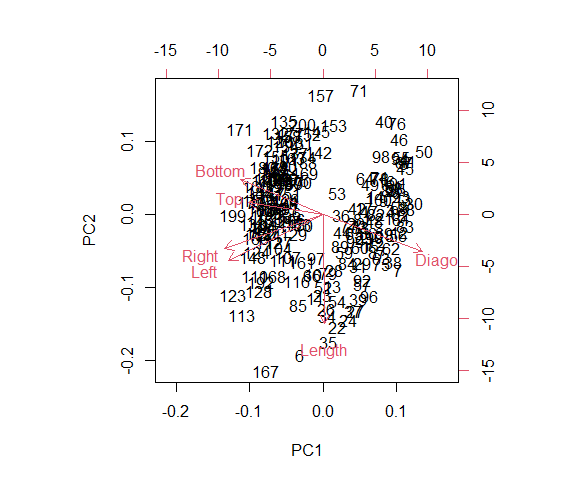
We can view the proportion of variation accounted for with each PC:

par(mfrow = c(1, 2))  
plot(PCs$sdev^2/sum(PCs$sdev^2),   
 xlab = "Principal Component",   
 ylab = "Proportion of Variance Explained",   
 ylim = c(0, 1),   
 type = 'b')  
plot(cumsum(PCs$sdev^2/sum(PCs$sdev^2)),   
 xlab = "Principal Component",  
 ylab = "Cum. Prop of Variance Explained",   
 ylim = c(0, 1),   
 type = 'b')



Let’s visualize our data with just the first two PCs.

biplot(PCs, cex = 1)



We could interpret the first PC as representing the diagonal measurement against the top, left, right, and bottom measurements.

The second PC then seems to represent the length in relation to the bottom, right, left, and diagonal measruements.

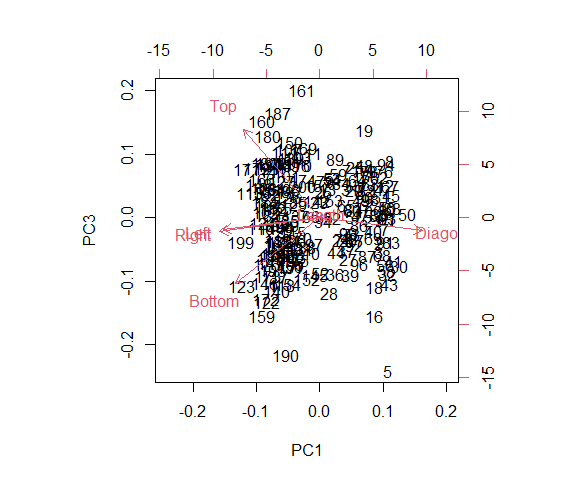
PCs

Standard deviations (1, .., p=6):  
[1] 1.7162629 1.1305237 0.9322192 0.6706480 0.5183405 0.4346031  
  
Rotation (n x k) = (6 x 6):  
 PC1 PC2 PC3 PC4 PC5 PC6  
Length 0.006987029 -0.81549497 0.01768066 0.5746173 -0.0587961 0.03105698  
Left -0.467758161 -0.34196711 -0.10338286 -0.3949225 0.6394961 -0.29774768  
Right -0.486678705 -0.25245860 -0.12347472 -0.4302783 -0.6140972 0.34915294  
Bottom -0.406758327 0.26622878 -0.58353831 0.4036735 -0.2154756 -0.46235361  
Top -0.367891118 0.09148667 0.78757147 0.1102267 -0.2198494 -0.41896754  
Diagonal 0.493458317 -0.27394074 -0.11387536 -0.3919305 -0.3401601 -0.63179849

We have effectively mapped the six variables in the dataset into two new features, and thus reduced the dimension by 4. Each observation (a six dimensional vector) is mapped to a two dimensional point in the new plot, this making visualization much easier.

Note: We could look at the first and third as well.

biplot(PCs, choices = c(1,3))



The third PC seems to be a top vs bottom comparison.

### PCA Recap

* Principal Components Analysis (PCA) is a dimension reduction technique
* PCA looks for *linear combinations* of variables that accounts for as much variability as possible
* Reduces dimension of data from to
* Choosing is subjective!
* Interpretation is subjective!
* Scaling is needed (most of the time) since units will vastly change results
* Can use as predictors in a regression (or other) model (PC Regression)