Untitled

Mohammed Ali

December 28, 2017

R Markdown

This to go through the case study from datacamp unsupervised learning course

Preparing the data

Use read.csv() function to download the CSV (comma-separated values) file containing the data from the URL provided. Assign the result to wisc.df.

```
url <- "http://s3.amazonaws.com/assets.datacamp.com/production/course_1903/datasets/WisconsinCancer.csv
# Download the data: wisc.df
wisc.df <- read.csv(file = url)
str(wisc.df)
```

```
## 'data.frame':    569 obs. of  33 variables:
##  $ id                      : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis               : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean             : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean            : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean               : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean         : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean        : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean          : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean     : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean           : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean  : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se               : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se              : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se            : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                 : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se           : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se          : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se            : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se       : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se             : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se    : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst            : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst           : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst         : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst              : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst        : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst       : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst         : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst    : num  0.265 0.186 0.243 0.258 0.163 ...
```

```
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
##  $ X                      : logi  NA NA NA NA NA NA ...
```

Use as.matrix() to convert the features of the data (in columns 3 through 32) to a matrix. Store this in a variable called wisc.data.

```
# Convert the features of the data: wisc.data
wisc.data <- as.matrix(wisc.df[, 3:32])
```

Assign the row names of wisc.data the values currently contained in the id column of wisc.df. While not strictly required, this will help you keep track of the different observations throughout the modeling process.

```
# Set the row names of wisc.data
row.names(wisc.data) <- wisc.df$id
```

Finally, set a vector called diagnosis to be 1 if a diagnosis is malignant ("M") and 0 otherwise. Note that R coerces TRUE to 1 and FALSE to 0.

```
# Create diagnosis vector
diagnosis <- as.numeric(wisc.df$diagnosis == "M")
```

Exploratory data analysis

The first step of any data analysis, unsupervised or supervised, is to familiarize yourself with the data.

The variables you created before, wisc.data and diagnosis, are still available in your workspace. Explore the data to answer the following questions:

- How many observations are in this dataset?

```
nrow(wisc.data)
```

```
## [1] 569
```

- How many variables/features in the data are suffixed with _mean?

```
length(grep("_mean", colnames(wisc.data)))
```

```
## [1] 10
```

- How many of the observations have a malignant diagnosis?

```
table(diagnosis)
```

```
## diagnosis
##   0   1
## 357 212
```

Performing PCA

The next step in your analysis is to perform PCA on wisc.data.

You saw in the last chapter that it's important to check if the data need to be scaled before performing PCA. Recall two common reasons for scaling data:

- The input variables use different units of measurement.
- The input variables have significantly different variances.

Check the mean and standard deviation of the features of the data to determine if the data should be scaled. Use the colMeans() and apply() functions like you've done before.

```
# Check column means and standard deviations
colMeans(wisc.data)
```

```
##             radius_mean            texture_mean          perimeter_mean
##            1.412729e+01            1.928965e+01            9.196903e+01
##               area_mean         smoothness_mean        compactness_mean
##            6.548891e+02            9.636028e-02            1.043410e-01
##          concavity_mean     concave.points_mean           symmetry_mean
##            8.879932e-02            4.891915e-02            1.811619e-01
##  fractal_dimension_mean               radius_se              texture_se
##            6.279761e-02            4.051721e-01            1.216853e+00
##            perimeter_se                 area_se           smoothness_se
##            2.866059e+00            4.033708e+01            7.040979e-03
##          compactness_se            concavity_se       concave.points_se
##            2.547814e-02            3.189372e-02            1.179614e-02
##             symmetry_se    fractal_dimension_se            radius_worst
##            2.054230e-02            3.794904e-03            1.626919e+01
##           texture_worst         perimeter_worst              area_worst
##            2.567722e+01            1.072612e+02            8.805831e+02
##        smoothness_worst       compactness_worst         concavity_worst
##            1.323686e-01            2.542650e-01            2.721885e-01
##    concave.points_worst          symmetry_worst fractal_dimension_worst
##            1.146062e-01            2.900756e-01            8.394582e-02
```

```
apply(wisc.data, 2, sd)
```

```
##             radius_mean            texture_mean          perimeter_mean
##            3.524049e+00            4.301036e+00            2.429898e+01
##               area_mean         smoothness_mean        compactness_mean
##            3.519141e+02            1.406413e-02            5.281276e-02
##          concavity_mean     concave.points_mean           symmetry_mean
##            7.971981e-02            3.880284e-02            2.741428e-02
##  fractal_dimension_mean               radius_se              texture_se
##            7.060363e-03            2.773127e-01            5.516484e-01
##            perimeter_se                 area_se           smoothness_se
##            2.021855e+00            4.549101e+01            3.002518e-03
##          compactness_se            concavity_se       concave.points_se
##            1.790818e-02            3.018606e-02            6.170285e-03
##             symmetry_se    fractal_dimension_se            radius_worst
##            8.266372e-03            2.646071e-03            4.833242e+00
##           texture_worst         perimeter_worst              area_worst
##            6.146258e+00            3.360254e+01            5.693570e+02
##        smoothness_worst       compactness_worst         concavity_worst
##            2.283243e-02            1.573365e-01            2.086243e-01
##    concave.points_worst          symmetry_worst fractal_dimension_worst
##            6.573234e-02            6.186747e-02            1.806127e-02
```

Execute PCA on the wisc.data, scaling if appropriate, and assign the model to wisc.pr.

```
# Execute PCA, scaling if appropriate: wisc.pr
wisc.pr <- prcomp(wisc.data, scale=TRUE, center=TRUE)
```

Inspect a summary of the results with the summary() function.

```r
# Look at summary of results
summary(wisc.pr)
```

```
## Importance of components%s:
##                            PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880
## Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025
## Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759
##                            PC7     PC8    PC9    PC10   PC11    PC12
## Standard deviation     0.82172 0.69037 0.6457 0.59219 0.5421 0.51104
## Proportion of Variance 0.02251 0.01589 0.0139 0.01169 0.0098 0.00871
## Cumulative Proportion  0.91010 0.92598 0.9399 0.95157 0.9614 0.97007
##                           PC13    PC14    PC15    PC16    PC17    PC18
## Standard deviation     0.49128 0.39624 0.30681 0.28260 0.24372 0.22939
## Proportion of Variance 0.00805 0.00523 0.00314 0.00266 0.00198 0.00175
## Cumulative Proportion  0.97812 0.98335 0.98649 0.98915 0.99113 0.99288
##                           PC19    PC20   PC21    PC22    PC23   PC24
## Standard deviation     0.22244 0.17652 0.1731 0.16565 0.15602 0.1344
## Proportion of Variance 0.00165 0.00104 0.0010 0.00091 0.00081 0.0006
## Cumulative Proportion  0.99453 0.99557 0.9966 0.99749 0.99830 0.9989
##                           PC25    PC26    PC27    PC28    PC29    PC30
## Standard deviation     0.12442 0.09043 0.08307 0.03987 0.02736 0.01153
## Proportion of Variance 0.00052 0.00027 0.00023 0.00005 0.00002 0.00000
## Cumulative Proportion  0.99942 0.99969 0.99992 0.99997 1.00000 1.00000
```

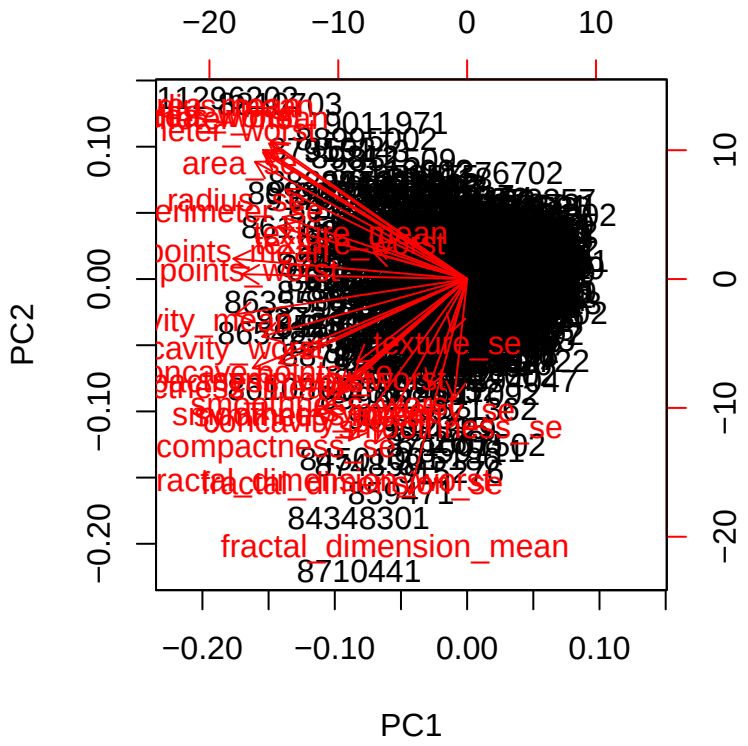
Interpreting PCA results

Now you'll use some visualizations to better understand your PCA model. You were introduced to one of these visualizations, the biplot, in an earlier chapter.

You'll run into some common challenges with using biplots on real-world data containing a non-trivial number of observations and variables, then you'll look at some alternative visualizations. You are encouraged to experiment with additional visualizations before moving on to the next exercise.

Create a biplot of the wisc.pr data. What stands out to you about this plot? Is it easy or difficult to understand? Why?

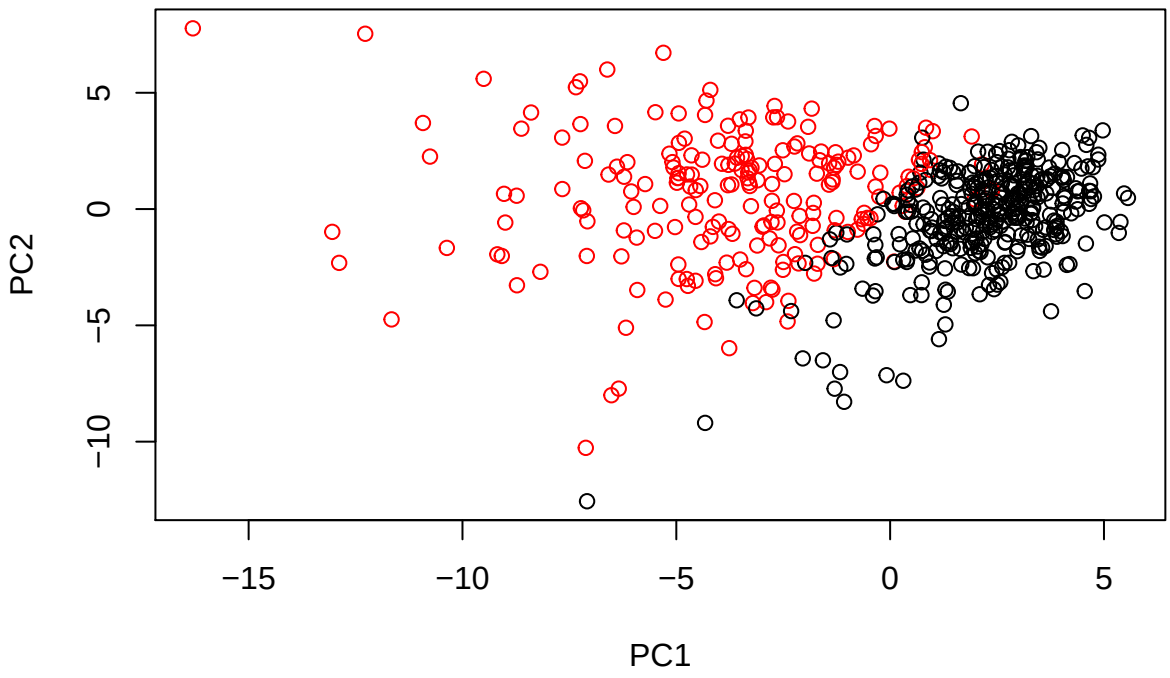
```r
# Create a biplot of wisc.pr
biplot(wisc.pr)
```



It seems not easy to read due to much data.

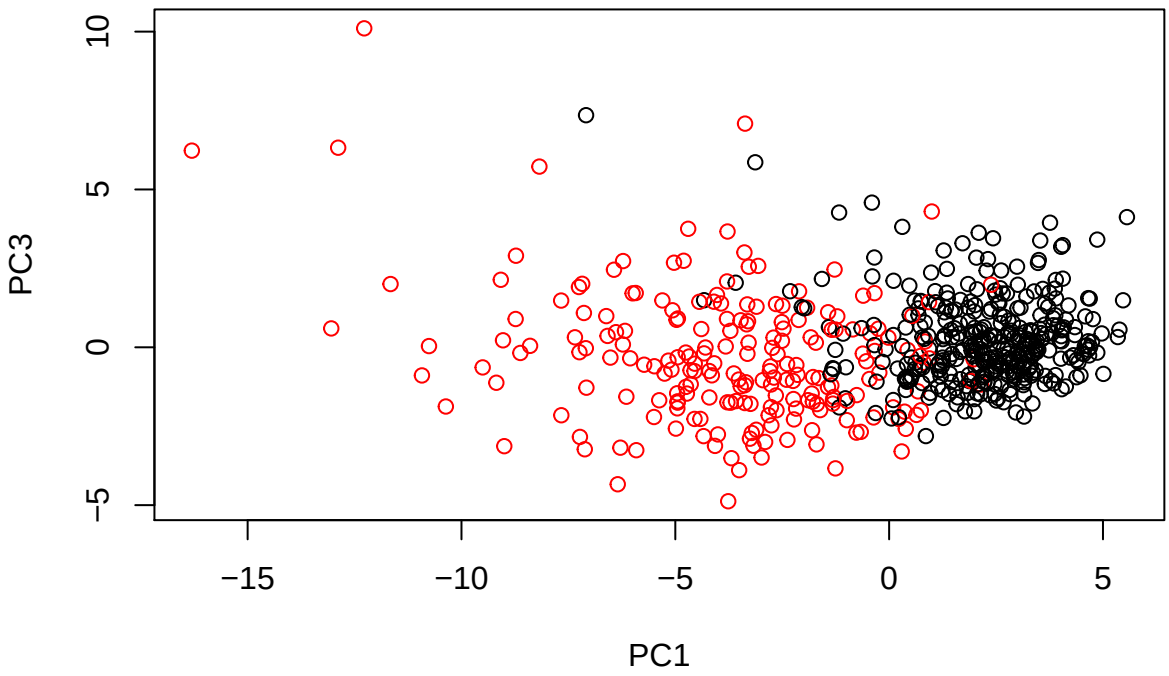
Execute the code to scatter plot each observation by principal components 1 and 2, coloring the points by the diagnosis.

```
# Scatter plot observations by components 1 and 2
plot(wisc.pr$x[, c(1, 2)], col = (diagnosis + 1),
     xlab = "PC1", ylab = "PC2")
```



Repeat the same for principal components 1 and 3. What do you notice about these plots?

```
# Repeat for components 1 and 3
plot(wisc.pr$x[, c(1, 3)], col = (diagnosis + 1),
     xlab = "PC1", ylab = "PC3")

# Do additional data exploration of your choosing below (optional)
plot(wisc.pr$x[, c(1, 3)], col = (diagnosis + 1),
     xlab = "PC1", ylab = "PC3")
```



Principal component 2 explains more variance in the original data than principal component 3, you can see that the first plot has a cleaner cut separating the two subgroups.

Variance explained

In this exercise, you will produce scree plots showing the proportion of variance explained as the number of principal components increases. The data from PCA must be prepared for these plots, as there is not a built-in function in R to create them directly from the PCA model.

As you look at these plots, ask yourself if there's an elbow in the amount of variance explained that might lead you to pick a natural number of principal components. If an obvious elbow does not exist, as is typical in real-world datasets, consider how else you might determine the number of principal components to retain based on the scree plot.

Calculate the variance of each principal component by squaring the sdev component of wisc.pr. Save the result as an object called pr.var.

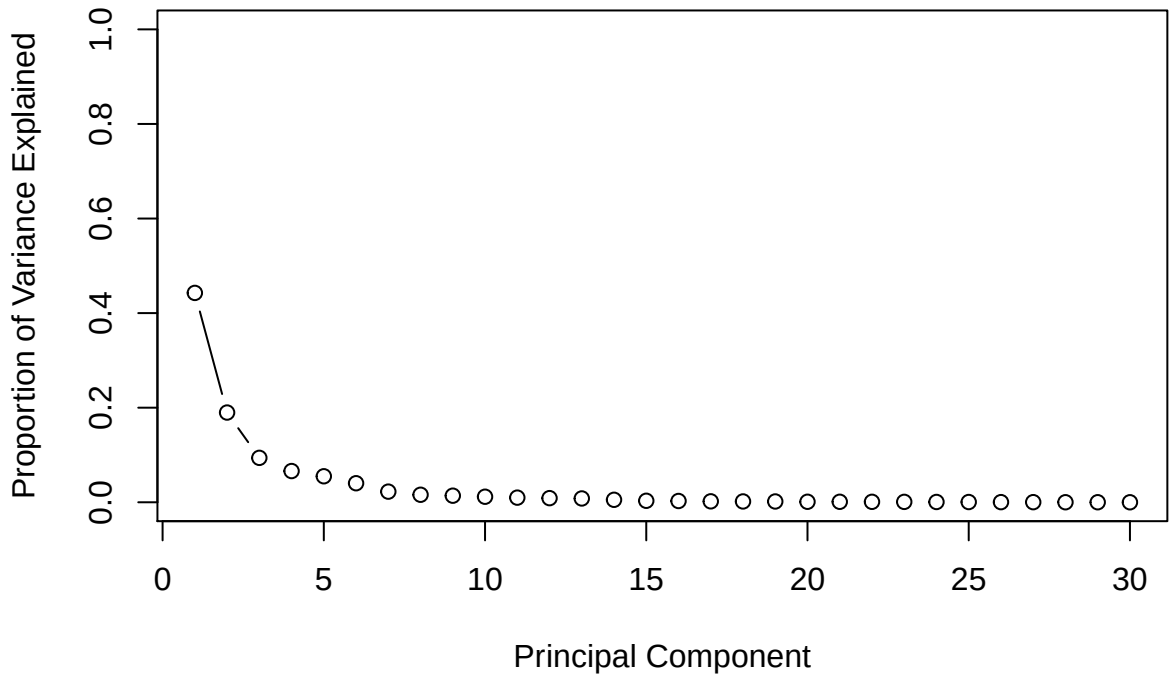
```
# Set up 1 x 2 plotting grid
par(mfrow = c(1, 2))

# Calculate variability of each component
pr.var <- wisc.pr$sdev * wisc.pr$sdev
```

Calculate the variance explained by each principal component by dividing by the total variance explained of all principal components. Assign this to a variable called pve.

```r
# Variance explained by each principal component: pve
pve <- pr.var/sum(pr.var)
```

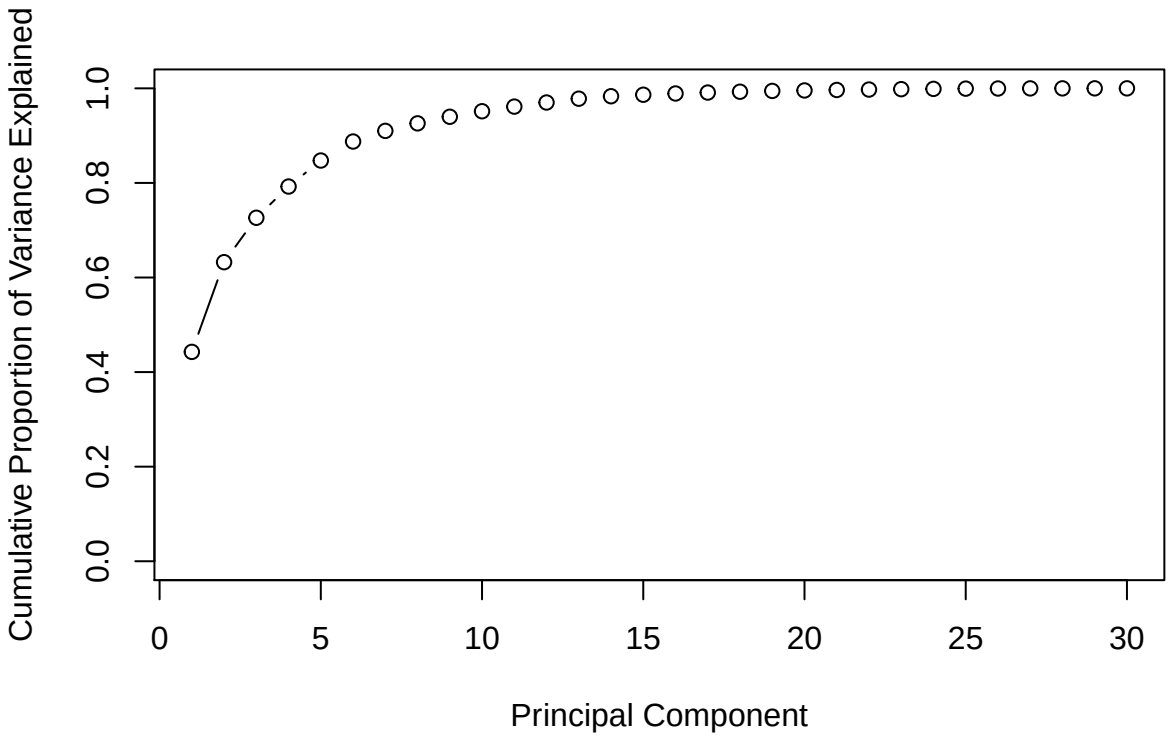
Create a plot of variance explained for each principal component.

```r
# Plot variance explained for each principal component
plot(pve, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0, 1), type = "b")
```



Using the cumsum() function, create a plot of cumulative proportion of variance explained.

```r
# Plot cumulative proportion of variance explained
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "b")
```



Before moving on, answer the following question: What is the minimum number of principal components needed to explain 80% of the variance in the data? Ans.5

Communicating PCA results

This exercise will check your understanding of the PCA results, in particular the loadings and variance explained. The loadings, represented as vectors, explain the mapping from the original features to the principal components. The principal components are naturally ordered from the most variance explained to the least variance explained.

For the first principal component, what is the component of the loading vector for the feature concave.points_mean?

```
wisc.pr$rotation[8,1]
```

```
## [1] -0.2608538
```

Hierarchical clustering of case data

The goal of this exercise is to do hierarchical clustering of the observations. Recall from Chapter 2 that this type of clustering does not assume in advance the number of natural groups that exist in the data.

As part of the preparation for hierarchical clustering, distance between all pairs of observations are computed. Furthermore, there are different ways to link clusters together, with single, complete, and average being the most common linkage methods.

Scale the wisc.data data and assign the result to data.scaled.

```
# Scale the wisc.data data: data.scaled
data.scaled <- scale(wisc.data)
```

Calculate the (Euclidean) distances between all pairs of observations in the new scaled dataset and assign the result to data.dist.

```
# Calculate the (Euclidean) distances: data.dist
data.dist <- dist(data.scaled)
```

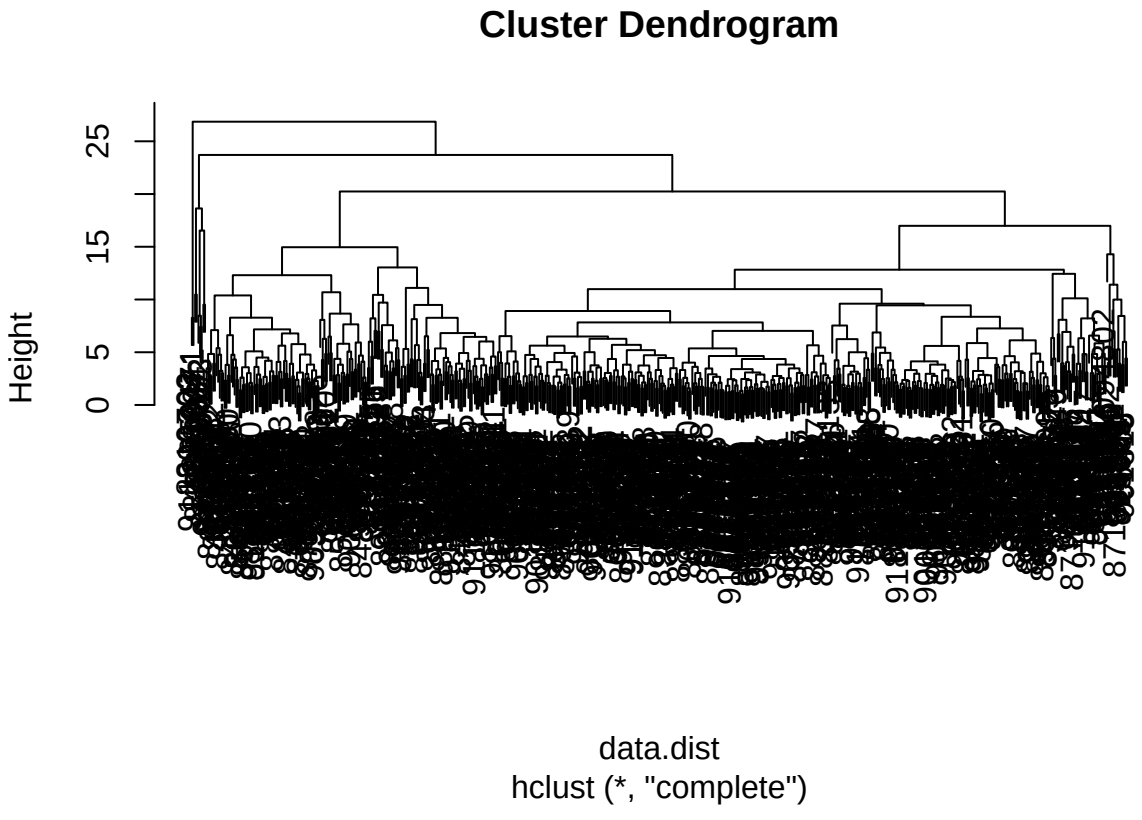
Create a hierarchical clustering model using complete linkage. Manually specify the method argument to hclust() and assign the results to wisc.hclust.

```
# Create a hierarchical clustering model: wisc.hclust
wisc.hclust <- hclust(data.dist, method = "complete")
```

Results of hierarchical clustering

Let's use the hierarchical clustering model you just created to determine a height (or distance between clusters) where a certain number of clusters exists. Ans. 20

```
plot(wisc.hclust)
```



Selecting number of clusters

In this exercise, you will compare the outputs from your hierarchical clustering model to the actual diagnoses. Normally when performing unsupervised learning like this, a target variable isn't available. We do have it with this dataset, however, so it can be used to check the performance of the clustering model.

When performing supervised learning—that is, when you're trying to predict some target variable of interest and that target variable is available in the original data—using clustering to create new features may or may not improve the performance of the final model. This exercise will help you determine if, in this case, hierarchical clustering provides a promising new feature.

Use cutree() to cut the tree so that it has 4 clusters. Assign the output to the variable wisc.hclust.clusters.

```
# Cut tree so that it has 4 clusters: wisc.hclust.clusters
wisc.hclust.clusters <- cutree(wisc.hclust, k=4)
```

Use the table() function to compare the cluster membership to the actual diagnoses.

```
# Compare cluster membership to actual diagnoses
table(diagnosis, wisc.hclust.clusters)
```

```
##           wisc.hclust.clusters
## diagnosis   1   2   3   4
##         0  12   2 343   0
##         1 165   5  40   2
```

Four clusters were picked after some exploration. Before moving on, you may want to explore how different numbers of clusters affect the ability of the hierarchical clustering to separate the different diagnoses.

k-means clustering and comparing results

As you now know, there are two main types of clustering: hierarchical and k-means.

In this exercise, you will create a k-means clustering model on the Wisconsin breast cancer data and compare the results to the actual diagnoses and the results of your hierarchical clustering model. Take some time to see how each clustering model performs in terms of separating the two diagnoses and how the clustering models compare to each other.

Create a k-means model on wisc.data, assigning the result to wisc.km. Be sure to create 2 clusters, corresponding to the actual number of diagnosis. Also, remember to scale the data and repeat the algorithm 20 times to find a well performing model.

```
# Create a k-means model on wisc.data: wisc.km
wisc.km <- kmeans(scale(wisc.data), centers = 2, nstart = 20)
```

Use the table() function to compare the cluster membership of the k-means model to the actual diagnoses contained in the diagnosis vector. How well does k-means separate the two diagnoses?

```
# Compare k-means to actual diagnoses
table(diagnosis, wisc.km$cluster)
```

```
##
## diagnosis   1   2
##         0  14 343
##         1 175  37
```

```
# Compare k-means to hierarchical clustering
table(wisc.hclust.clusters, wisc.km$cluster)
```

```
##
## wisc.hclust.clusters   1   2
##                    1 160  17
##                    2   7   0
##                    3  20 363
##                    4   2   0
```

Nice! Looking at the second table you generated, it looks like clusters 1, 2, and 4 from the hierarchical clustering model can be interpreted as the cluster 1 equivalent from the k-means algorithm, and cluster 3 can be interpreted as the cluster 2 equivalent.

Clustering on PCA results

Recall from earlier exercises that the PCA model required significantly fewer features to describe 80% and 95% of the variability of the data. In addition to normalizing data and potentially avoiding overfitting, PCA also uncorrelates the variables, sometimes improving the performance of other modeling techniques.

Let's see if PCA improves or degrades the performance of hierarchical clustering.

Using the minimum number of principal components required to describe at least 90% of the variability in the data, create a hierarchical clustering model with complete linkage. Assign the results to wisc.pr.hclust.

```
wisc.pr.hclust <- hclust(dist(wisc.pr$x[, 1:7]), method = "complete")
```

Cut this hierarchical clustering model into 4 clusters and assign the results to wisc.pr.hclust.clusters.

```
wisc.pr.hclust.clusters <- cutree(wisc.pr.hclust, k=4)
```

Using table(), compare the results from your new hierarchical clustering model with the actual diagnoses. How well does the newly created model with four clusters separate out the two diagnoses?

```
# Compare to actual diagnoses
table(diagnosis, wisc.hclust.clusters)
```

```
##          wisc.hclust.clusters
## diagnosis   1   2   3   4
##         0  12   2 343   0
##         1 165   5  40   2
```

```
table(diagnosis, wisc.pr.hclust.clusters)
```

```
##          wisc.pr.hclust.clusters
## diagnosis   1   2   3   4
##         0   5 350   2   0
##         1 113  97   0   2
```

How well do the k-means and hierarchical clustering models you created in previous exercises do in terms of separating the diagnoses? Again, use the table() function to compare the output of each model with the vector containing the actual diagnoses.

```
# Compare to k-means and hierarchical
table(wisc.km$cluster, diagnosis)
```

```
##    diagnosis
##       0   1
##   1  14 175
##   2 343  37
```