STAT167 Lab #9 - Spring 2025

Ethan Choi

2025/5/30

Contents

Discussion/Lab #9 instructions

This week, we will review classification evaluation metrics, and unsupervised learning methods, including K-means clustering and hierarchical clustering.

- First, download the `rmd` file from Canvas.
- Open this `rmd` file in RStudio and click `Knit -> Knit to PDF` to render it to PDF format. You need to have `LaTex` installed on the computer to render it to PDF format. If not, you can also render it to HTML format.
- Read this `rmd` file and the rendered `pdf`/`html` file side-by-side, to see how this document was generated!
- Be sure to play with this document! Change it. Break it. Fix it. The best way to learn R Markdown (or really almost anything) is to try, fail, then find out what you did wrong.
- Read over the example code and the output. If you have any questions about certain functions or parameters, it is the time to ask!
- There are some exercises through out this document. Replace **INSERT_YOUR_ANSWER** with your own answers. Knit the file, and check your results.

Please comment your R code thoroughly, and follow the R coding style guideline (https://google.github.io/styleguide/Rguide.xml). Partial credit will be deducted for insufficient commenting or poor coding styles.

Lab submission guideline

- After you completed all exercises, save your file to `FirstnameLastname-SID-lab9.rmd` and save the rendered pdf file to `FirstnameLastname-SID-lab9.pdf`. If you can not knit it to pdf, knit it to html first and then print/save it to pdf format.
- Submit **BOTH your source `rmd` file and the knitted `pdf` file** to **GradeScope**. Do NOT create a zip file.
- You can submit multiple times, you last submission will be graded.

Install necessary packages

Note that you only need to install each package once. Then you can comment out the following installation lines.

```
#install.packages("tidyverse")
#install.packages("ggdendro")
```

Load necessary packages

```
library(tidyverse) # for `ggplot2`, `dplyr`, `tibble`, and more
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
```

```
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
library(ISLR) # for `Default` dataset
library(boot) # for cv.glm()
library(plotROC) # for ROC curve
library(ggdendro) # for plotting dendrogram
```

Set the random seed

```
# set the random seed so that the analysis is reproducible
set.seed(232)
```

Lecture Review - Evaluation of Classification Methods

The `Default` data set

The `Default` data set (included in the `ISLR` library) contains 10,000 credit card customer information.

```
?Default # full documentation
## starting httpd help server ... done
glimpse(Default)
## Rows: 10,000
## Columns: 4
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, No, No, No, No~
## $ student <fct> No, Yes, No, No, No, Yes, No, Yes, No, No, Yes, Yes, No, No, N~
## $ balance <dbl> 729.5265, 817.1804, 1073.5492, 529.2506, 785.6559, 919.5885, 8~
## $ income  <dbl> 44361.625, 12106.135, 31767.139, 35704.494, 38463.496, 7491.55~
```

We can call `glm()` with the argument `family = binomial()` to perform logistic regression to predict `default`.

```
logit.fit.all <- glm(default ~ ., family = binomial(), data = Default)
summary(logit.fit.all)
##
## Call:
## glm(formula = default ~ ., family = binomial(), data = Default)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
## studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
## balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
```

```
## income       3.033e-06  8.203e-06   0.370  0.71152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.5  on 9996  degrees of freedom
## AIC: 1579.5
##
## Number of Fisher Scoring iterations: 8
```

Misclassification error rate

To evaluate the classification accuracy, we can calculate the **misclassification error rate** on the training (or validation/test) data set.

$$\text{Error}_{\text{train}} = \frac{1}{n_{\text{train}}} \sum_{i \in \text{train}} I(y_i \neq \hat{y}_i)$$

$$\text{Error}_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{test}} I(y_i \neq \hat{y}_i)$$

Cross-Validation misclassification error rate

Similarly as we learned in regression evaluation, we can also apply cross-validation to evaluate classification models.

In the `cv.glm()` help page, it provided the following cost function for logistic regression.

```
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
```

Since logistic regression is a generalized linear regression model, we can call `cv.glm()` (included in the `boot` library) to calculate the (weighted) $\text{ErrorRate}_{\text{CV}}$.

$$\text{ErrorRate}_{\text{CV.weighted}} = \sum_{k=1}^{K} \frac{n_k}{n} \text{ErrorRate}_k$$

where n_k is the number of observations in fold k and ErrorRate_k is the misclassification error rate estimated from fold k.

```
cv.glm(Default, logit.fit.all, cost=cost, K=10)$delta[1]
## [1] 0.0269
```

TP, FP, TN, FN

In addition to misclassification errors, we can also define true positive (TP), true negative (TN), false positive (FP), and false negative (FN) using the following table.

	Predicted/False	Predicted/True
Observed/False	True Negative (TN)	False Positive (FP)
Observed/True	False Negative (FN)	True Positive (TP)

```
# extract conditional prob. P(default="Yes" | balance, income, student)
logit.fit.prob <- predict(logit.fit.all, type = "response")
# Bayes rule
logit.fit.class <- ifelse(logit.fit.prob > 0.5, "Yes", "No") %>% as.factor()

confusion.matrix <- table(Default$default, logit.fit.class)
confusion.matrix
##      logit.fit.class
##         No  Yes
##   No  9627   40
##   Yes  228  105
```

Exercise #1

How many FPs are there in the `logit.fit.class` predictions?

40

Sensitivity vs Specificity

To evaluate classification outcome, we can also use other classification metrics include:

- **Sensitivity / Recall / Power / True Positive Rate (TPR)**

$$= \frac{\text{\# observations correctly classified to have the event}}{\text{\# observations that had the event}}$$
$$= \frac{TP}{TP + FN} = \frac{TP}{P}$$

- **Specificity / True Negative Rate (TNR)**

$$= \frac{\text{\# observations correctly classified as non-events}}{\text{\# observations that did not have the event}}$$
$$= \frac{TN}{TN + FP} = \frac{TN}{N}$$

- **False Positive Rate (FPR)** = 1 - Specificity

Exercise #2

Calculate sensitivity and specificity of the `logit.fit.class` predictions.

```r
TP <- confusion.matrix["Yes", "Yes"]
TN <- confusion.matrix["No", "No"]
FP <- confusion.matrix["No", "Yes"]
FN <- confusion.matrix["Yes", "No"]

sensitivity <- TP / (TP + FN)
specificity <- TN / (TN + FP)

sensitivity
```

```
## [1] 0.3153153
```

```r
specificity
```

```
## [1] 0.9958622
```

Receiver operating characteristic (ROC) curve

To visualize the tradeoff between sensitivity and specificity, we can plot a ROC curve.

- x-axis: False Positive Rate (FPR) = 1 - Specificity = Type I error
- y-axis: True Positive Rate (TPR) = Sensitivity = 1 - Type II error

The `plotROC` package is helpful for generating ROC curves.

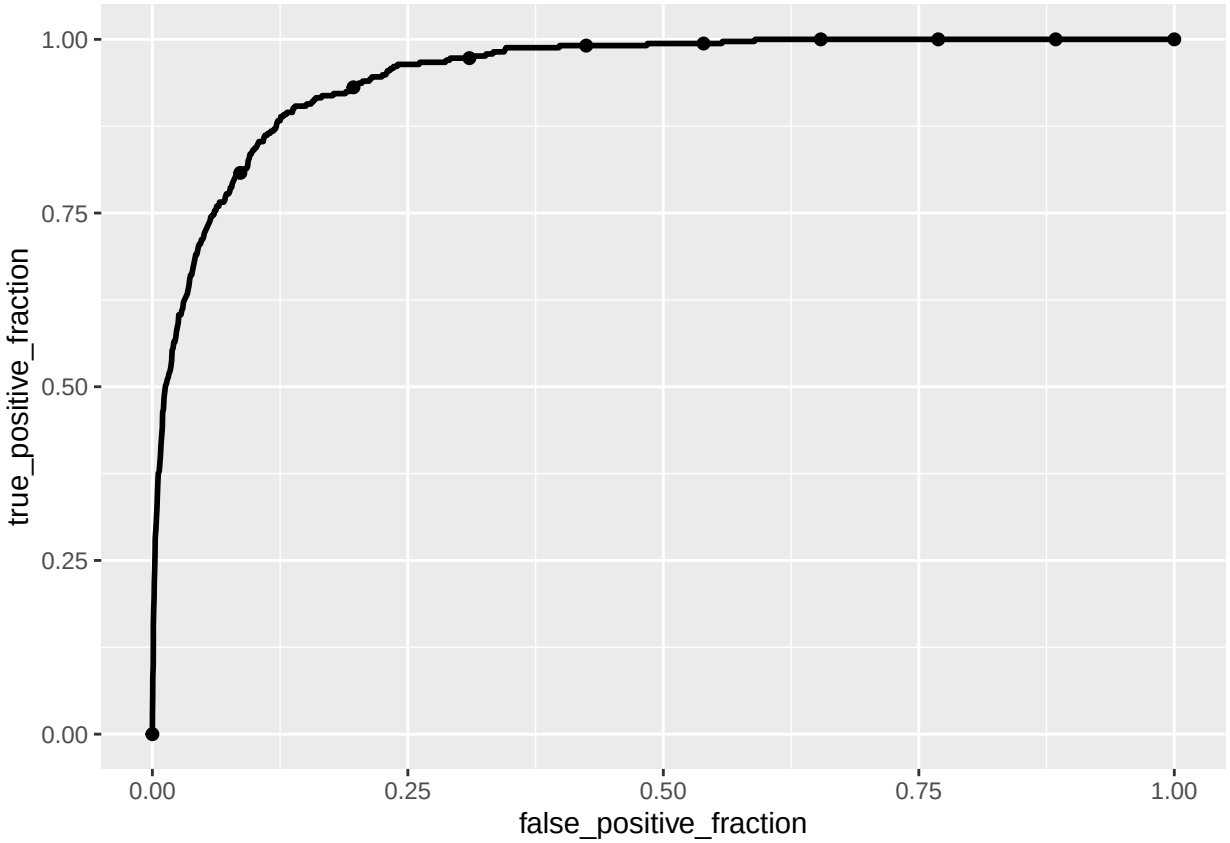
Here we can plot the ROC curve for the `logit.fit.all` predictions.

```r
library(plotROC)

# extract conditional prob. P(default="Yes" | balance, income, student)
logit.fit.prob <- predict(logit.fit.all, type = "response")

roc.df <- tibble(observed = Default$default,
                 predicted = logit.fit.prob)

ggplot(data = roc.df, mapping = aes(d = observed, m = predicted)) +
  geom_roc(labels=F)
## Warning in verify_d(data$d): D not labeled 0/1, assuming No = 0 and Yes = 1!
```



Lecture Review - K-Means Clustering

Goal: Find a partition of the data $\{C_1, \cdots, C_k\}$ that minimize the sum of **within-cluster variations** (WCVs)

$$\sum_{k=1}^{K} WCV(C_k)$$

Typically, we use the sum of all pairwise squared **Euclidean distances** between the observations in C_k to define $WCV(C_k)$.

$$WCV(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|\boldsymbol{x_i} - \boldsymbol{x_{i'}}\|_2^2 = 2 \sum_{i \in C_k} \|\boldsymbol{x_i} - \bar{\boldsymbol{x}}_{\boldsymbol{k}}\|_2^2$$

where $|C_k|$ denotes the number of observations in cluster k, and $\bar{\boldsymbol{x}}_{\boldsymbol{k}} = \frac{1}{|C_k|} \sum_{i \in C_k} \boldsymbol{x_i}$ is the average of all the points in cluster k, i.e., the cluster mean/centroid.

K-means clustering algorithm

1. Start by randomly partitioning the observations into K clusters (that is, randomly assign each observation to one of K clusters)

2. Until the clusters stop changing, **repeat**:

    - For each cluster C_k, compute the cluster centroid $\bar{\boldsymbol{x}}_{\boldsymbol{k}}$
    - Assign each observation to the cluster whose centroid is the closest (where "closest" is defined using Euclidean distance)

`kmeans()` implementation

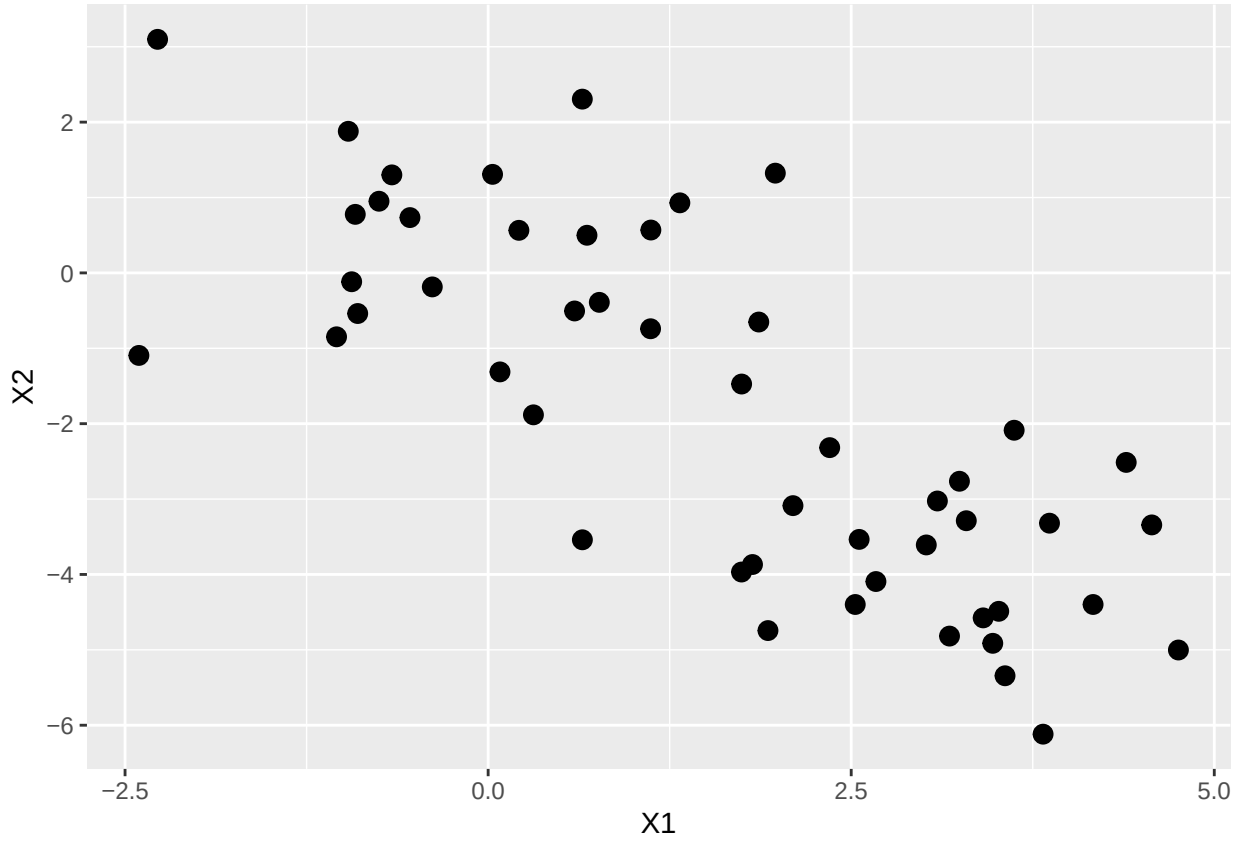
Computationally, we can use `stats::kmeans()` function to apply K-means clustering.

First we generate our toy example data.

```
# generate the example data
set.seed(232)
data <- matrix(rnorm(50*2), ncol = 2)
colnames(data) <- c("X1","X2")
data[1:25, 1] <- data[1:25, 1]+3
data[1:25, 2] <- data[1:25, 2]-4
data.tb <- as_tibble(data)
data.tb
## # A tibble: 50 x 2
##       X1    X2
##    <dbl> <dbl>
##  1  4.75 -5.00
##  2  1.74 -3.97
##  3  3.41 -4.58
##  4  3.25 -2.76
##  5  3.02 -3.61
##  6  3.52 -4.49
##  7  3.47 -4.91
##  8  1.82 -3.87
##  9  3.18 -4.82
## 10  1.93 -4.74
## # i 40 more rows
```

The first 25 points are centered at (3, -4) and the second 25 points are centered at (0, 0).

```
# visualize the example data
ggplot(data = data.tb, mapping = aes(x = X1, y = X2)) +
  geom_point(size = 3)
```



Next, let's call the `kmeans()` function.

```
?kmeans
# k-means clustering with k=3
km.out <- kmeans(data.tb, centers = 3, nstart = 20)
km.out
## K-means clustering with 3 clusters of sizes 14, 25, 11
##
## Cluster means:
##           X1         X2
## 1 -0.7775722  0.7236146
## 2  3.0927397 -3.8865841
## 3  1.0527951 -0.3307282
##
## Clustering vector:
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 3 1 3 3 3 3 1 3 1 1
## [39] 1 1 3 3 3 1 3 1 3 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 28.07858 46.93877 14.80215
##  (between_SS / total_SS =  80.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```r
# add the k-means results as a new column to data.tb
data.km <- add_column(data.tb, km.cluster = as.factor(km.out$cluster))
data.km
## # A tibble: 50 x 3
##       X1    X2 km.cluster
##    <dbl> <dbl> <fct>
##  1  4.75 -5.00 2
##  2  1.74 -3.97 2
##  3  3.41 -4.58 2
##  4  3.25 -2.76 2
##  5  3.02 -3.61 2
##  6  3.52 -4.49 2
##  7  3.47 -4.91 2
##  8  1.82 -3.87 2
##  9  3.18 -4.82 2
## 10  1.93 -4.74 2
## # i 40 more rows
```

```r
# visualize the results
ggplot(data = data.km, mapping = aes(x = X1, y = X2)) +
  geom_point(aes(color = km.cluster), size = 3) +
  ggtitle("K-Means Clustering Results with K = 3")
```

Exercise #3

(a) Read the helper page of `kmeans()`, what is the `nstart` argument for? How to extract the total within-cluster sum of squares from the `km.out` results?

nstart is the amount of random configs that the given algo will run. To extract the total within cluster SSQ from the km.out results, we use km.out$tot.withinss

(b) Run `kmeans()` with `nstart = 1` and `nstart = 20`, separately. Are the clustering results the same? What are the corresponding total within-cluster sum of squares?

```
km.out1 <- kmeans(data.tb, centers = 3, nstart = 1)
km.out20 <- kmeans(data.tb, centers = 3, nstart = 20)

km.out1$tot.withinss
```

```
## [1] 92.03837
```

```
km.out20$tot.withinss
```

```
## [1] 89.8195
```

Not the same. nstart = 1 yields 96.67098, nstsart = 20 yields 89.8195

Lecture Review - Hierarchical Clustering

Hierarchical clustering is an alternative clustering approach that does not require a pre-specified choice of K, and which provides a **deterministic** answer (no randomness).

Here we focus on **bottom-up** or **agglomerative** hierarchical clustering.

Hierarchical clustering algorithm (bottom-up / agglomerative approach)

1. Start with each point in its own cluster.
2. Identify the two closest clusters. Merge them.
3. Repeat until all points are in a single cluster

`hclust()` implementation

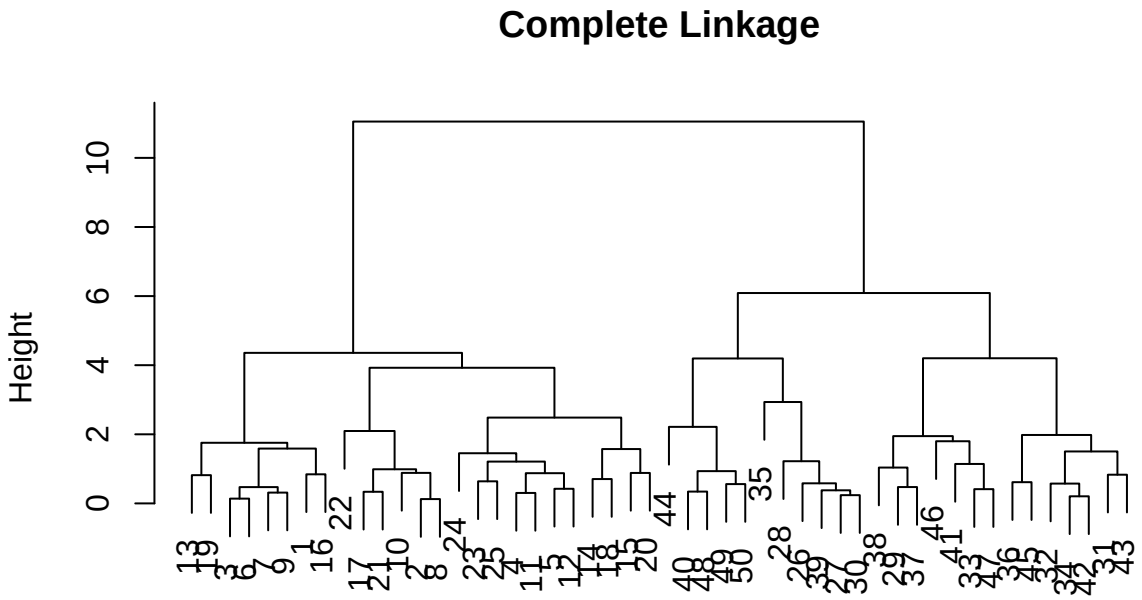
Computationally, we can use `stats::hclust()` function to apply the hierarchical clustering.

```
# hclust() helper page
?hclust

# Hierarchical Clustering
hc.complete <- hclust(d = dist(data.tb), method = "complete")
hc.complete
```

```
##
## Call:
## hclust(d = dist(data.tb), method = "complete")
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 50
```

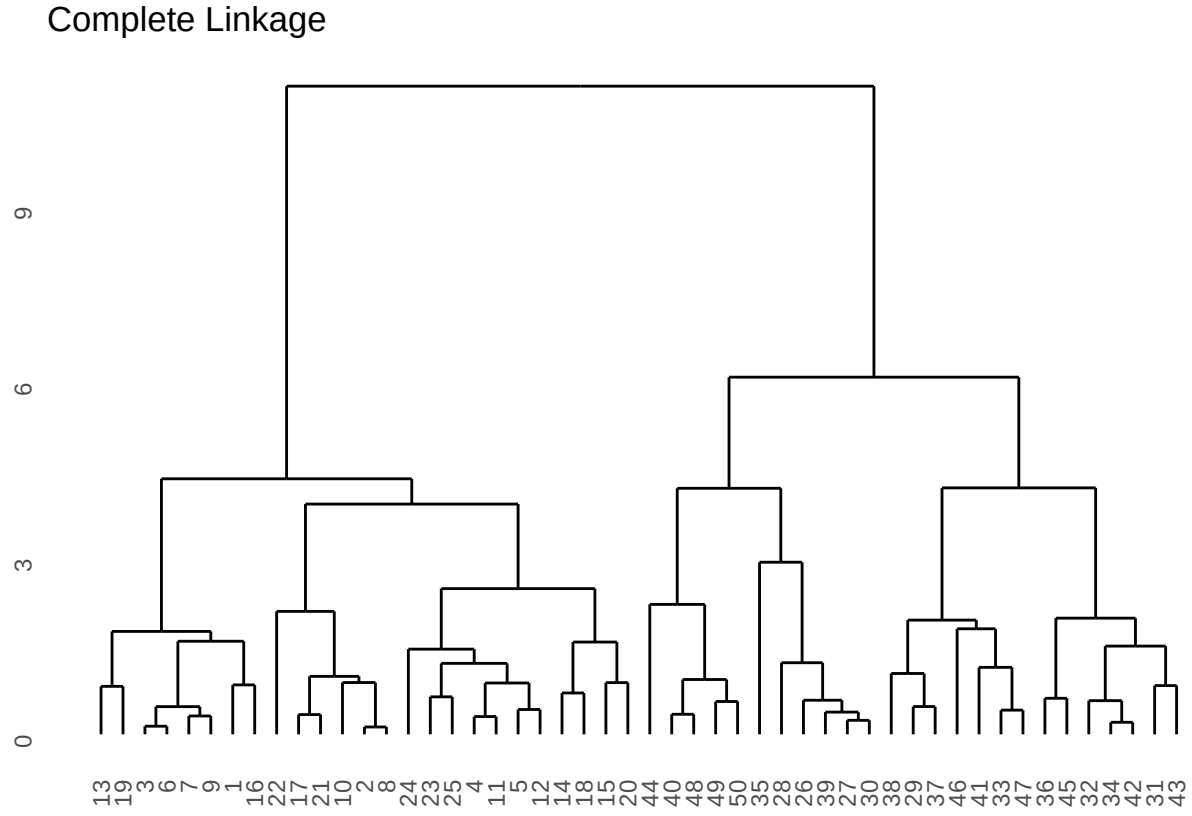
```
# base R plotting
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "")
```



ggplot2 doesn't have a geom function for dendrogram, but we can use the ggdendro package to plot dendrogram within the ggplot2 framework.

More information about ggdendro can be found at https://cran.r-project.org/web/packages/ggdendro/vignettes/ggdendro.html

```
library(ggdendro)
ggdendrogram(data = hc.complete) +
  ggtitle("Complete Linkage")
```

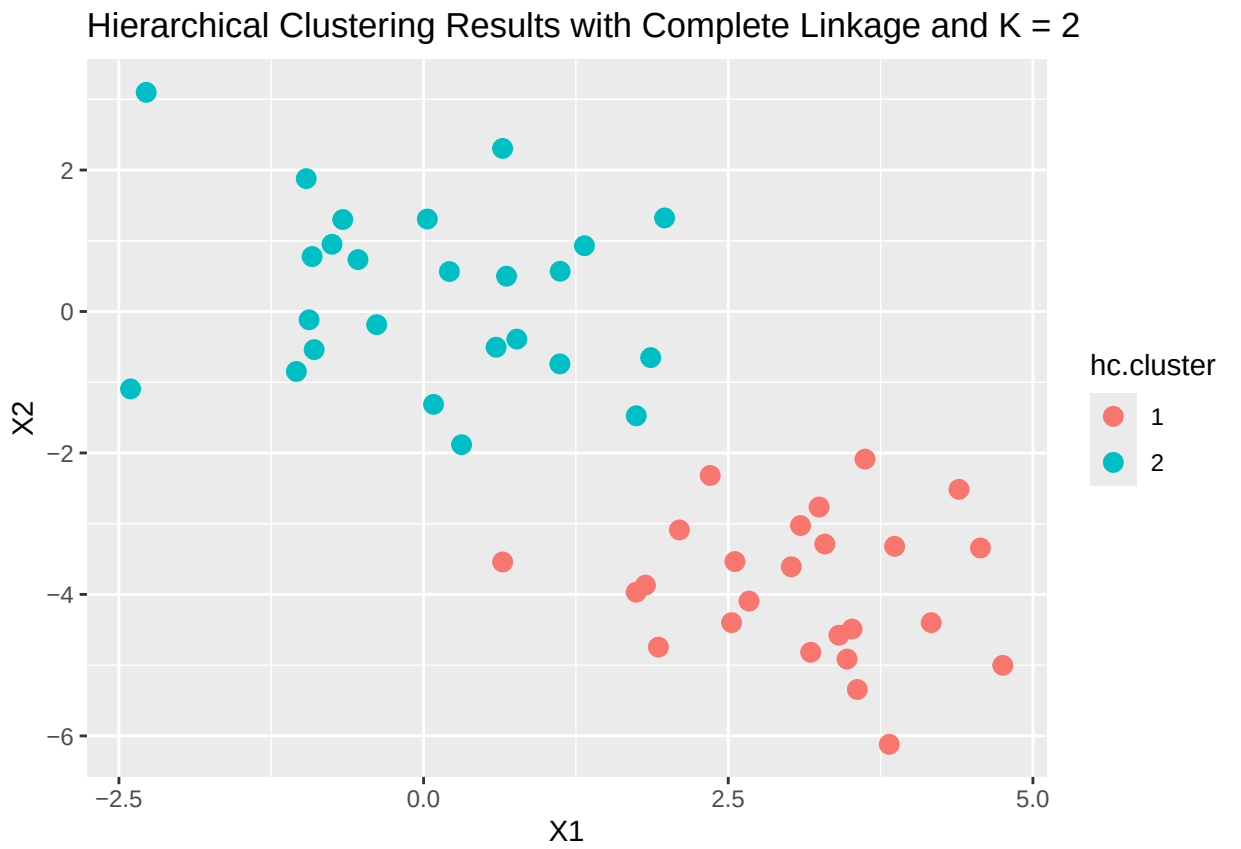


After obtaining the dendrogram tree, we can cut the tree to obtain the clusters.

```
hc.cl <- cutree(tree = hc.complete, k = 2)

# add the hierarchical clustering results as a new column to data.tb
data.hc <- add_column(data.tb, hc.cluster = as.factor(hc.cl))
data.hc
## # A tibble: 50 x 3
##       X1    X2 hc.cluster
##    <dbl> <dbl> <fct>
##  1  4.75 -5.00 1
##  2  1.74 -3.97 1
##  3  3.41 -4.58 1
##  4  3.25 -2.76 1
##  5  3.02 -3.61 1
##  6  3.52 -4.49 1
##  7  3.47 -4.91 1
##  8  1.82 -3.87 1
##  9  3.18 -4.82 1
## 10  1.93 -4.74 1
## # i 40 more rows
```

```
# visualize the results
ggplot(data = data.hc, mapping = aes(x = X1, y = X2)) +
  geom_point(aes(color = hc.cluster), size = 3) +
  ggtitle("Hierarchical Clustering Results with Complete Linkage and K = 2")
```



Linkage function

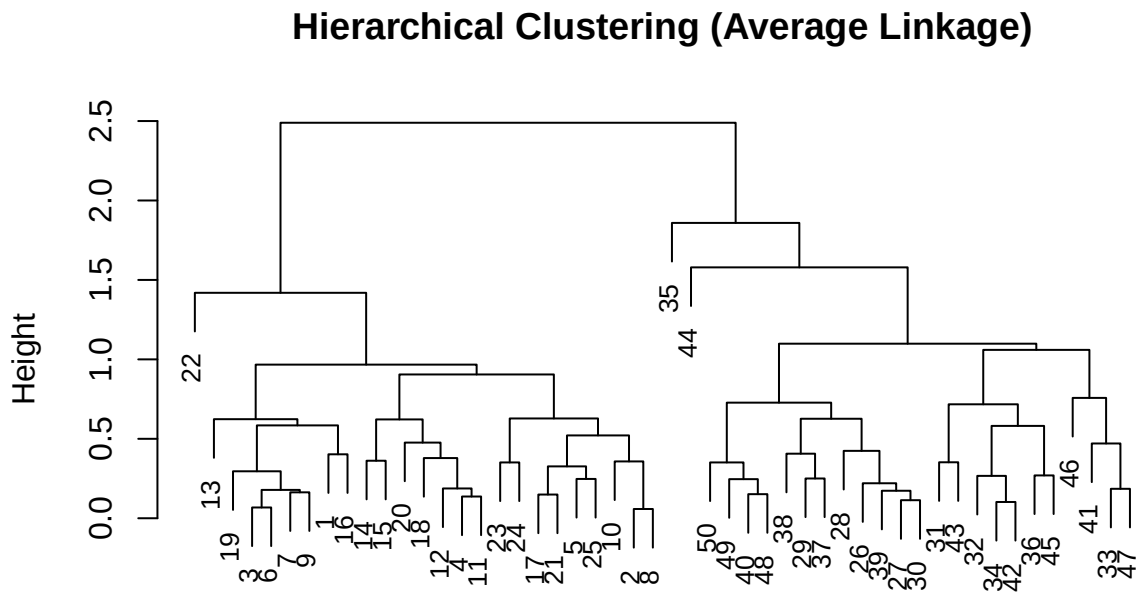
In the lecture, we have learned different linkage functions.

Linkage	Description
Single	Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **smallest** of these dissimilarities.
Complete	Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **largest** of these dissimilarities.
Average	Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **average** of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*.

Exercise #4

(a) Run the hierarchical clustering on `data.tb` using the **average linkage** function. Then visualize the dendrogram.

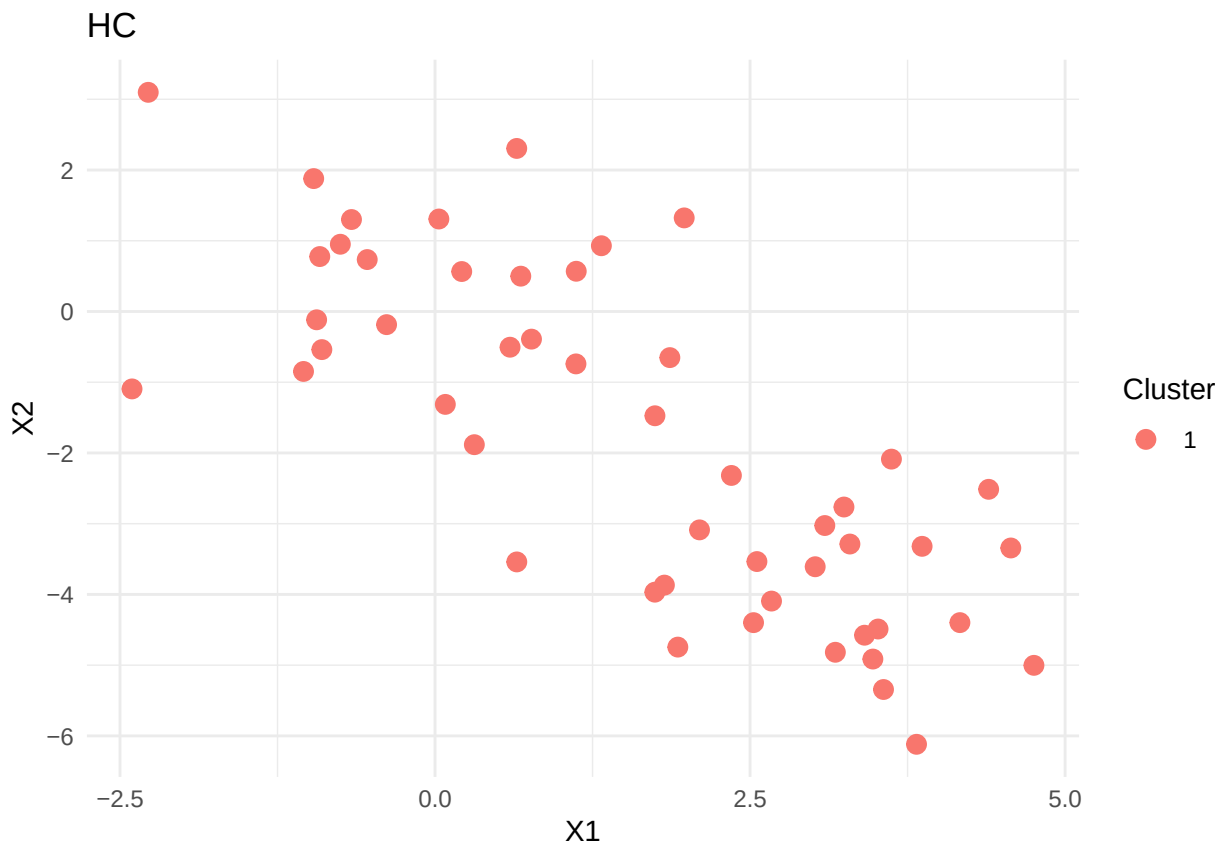
```r
data.scale <- scale(data.tb)
dist.data <- dist(data.scale)
hc.average <- hclust(dist.data, method = "average")
plot(hc.average, main = "Hierarchical Clustering (Average Linkage)", xlab = "", sub = "", cex = 0.8)
```



(b) Cut the dendrogram using the **`h = 2.5`** argument (not the `k` argument). Plot the sample data again and color label your clustering results. How many clusters do you get?

```r
hc.cluster <- cutree(hc.average, h = 2.5)
data.hc <- data.tb
data.hc$cluster <- factor(hc.cluster)

ggplot(data.hc, aes(x = X1, y = X2, color = cluster)) +
  geom_point(size = 3) +
  labs(title = "HC", color = "Cluster") +
  theme_minimal()
```




```
table(hc.cluster)
```

```
## hc.cluster
##  1
## 50
```

```
length(unique(hc.cluster))
```

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

```
#1 cluster, 50 in
```