Introduction to Unsupervised Learning

36-600

Fall 2021

The Setting

The setting for *unsupervised learning* is that you have a collection of p measurements (recorded in columns of a data frame) for each of n objects (recorded in rows of a data frame). The term "unsupervised" simply means that none of the variables is a response variable.

One can think of unsupervised learning as being an extension of EDA. In EDA, the goal is to visualize *projected* data to build intuition and to visually assess potential associations between variables. In unsupervised learning, we implement statistical algorithms to uncover potential structure in the data in their native space…e.g., separate clusters of data that represent naturally arising groups that you can identify and describe.

Sounds good. However, the main, overriding issue with unsupervised learning is that *there are no universally accepted mechanisms for model assessment or selection, i.e., there is no unique right answer!*

Digression: Similarity

Unsupervised learning relies on notions of *similarity*: how similar or dissimilar are two data? This dictates whether these data are to be drawn into one group or cluster, or not.

In the wider world of statistics and machine learning, there are many, many ways to quantify similarity. (Earth Mover's distance, anyone?) Here we will focus on the most intuitive, the L2-norm, much better known as the Euclidean distance:

$$d_{ij} = \sqrt{(X_{i1} - X_{j1})^2 + \cdots + (X_{ip} - X_{jp})^2}$$

Here, i and j are the indices for two data (i.e., the indices for two rows in a data frame) and X_1 through X_p represent the p measurements associated with each datum.

To compute pairwise distances in `R`, use `dist()` (which by default assume the Euclidean distance). Note that the output is a (symmetric) matrix, so *beware*: if your data frame has more than about 20,000 rows, your computer may have insufficient memory to store the matrix! (There are other, less-memory-intensive methods for computing pairwise distances for big datasets, but they are beyond the scope of today's notes.)

Clustering

What is clustering?

- It is the partitioning of data into homogeneous subgroups.

What is the goal of clustering?

- To define clusters for which the within-cluster variation is relatively small.

The what-now?

- Because we know the distances between each datum, we can determine the average squared distance between data *within defined clusters* and compare that to the same measure between *all data*. In a perfect world, the ratio of the first number to the second number is small! (Meaning that we've found small tight clusters that lie far apart.)

What are the caveats?

- Since clustering methods involve the computation of distances, *they are not applicable to categorical data*! If your three columns are `height`, `weight`, and `favorite ice cream flavor`, you'll have to either ignore the third column entirely, or run clustering for chocolate and vanilla lovers separately.
- In clustering methods, *all* data are forced into clusters. This may not be optimal. Thus there are algorithms like Gaussian mixture modeling, which we will cover next time.
- Any method that relies on distances will be impacted by units. For instance, if we have two columns with units of miles and units of millimeters, it stands to reason that in most situations the differences between data in millimeters will *dominate* the Euclidean distance.

Standardization

It is common practice to mitigate the potential biasing effect of units by *standardizing* the data within each column of the data frame:

$$X \to \frac{X - \bar{X}}{s_X},$$

where $\bar{X}$ and s_X are the sample mean and sample standard deviation of the data, respectively. (As a reminder, the functions to compute these quantities are `mean()` and `sd()`, but you don't actually need these here.) The distribution of standardized data could be anything, but the mean and the standard deviation of that distribution will be 0 and 1, respectively.

To standardize the data in columns of a data frame, simply use the `scale()` function:

```
df.scaled <- scale(df)
```

By default, it knows to treat the data frame one column at a time.

K-Means Clustering

The algorithm for K-means clustering is straightforward:

Algorithm 10.1 *K-Means Clustering*

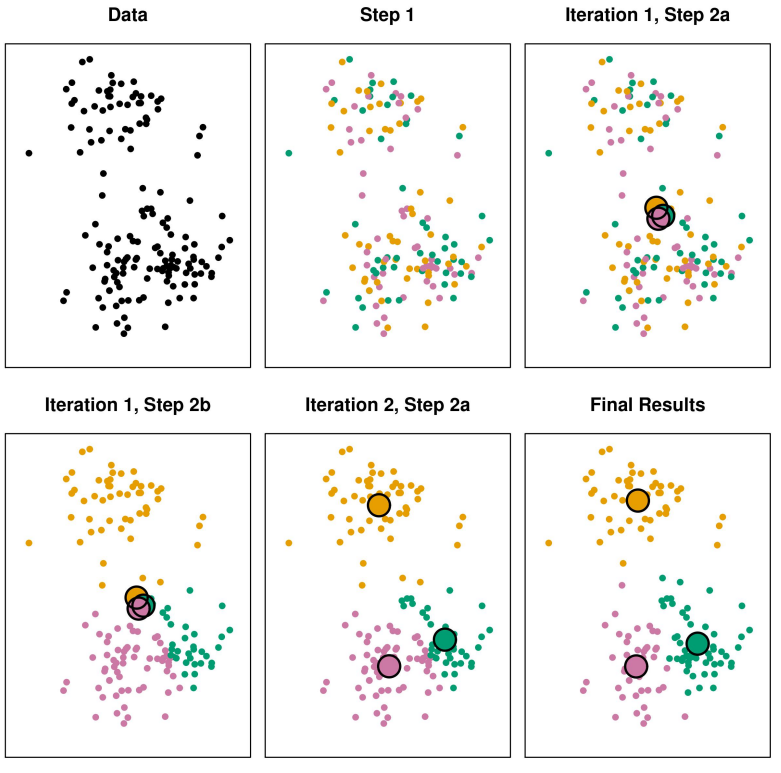
1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

(See the picture on the next slide for more intuition about what the algorithm is doing.)

Note the following:

- As stated above, there is no universally accepted metric that leads one to conclude that a particular value of K is the optimal one.
- Your results can change from run to run unless you explicitly set the random number seed immediately before calling `kmeans()`.

K-Means Clustering

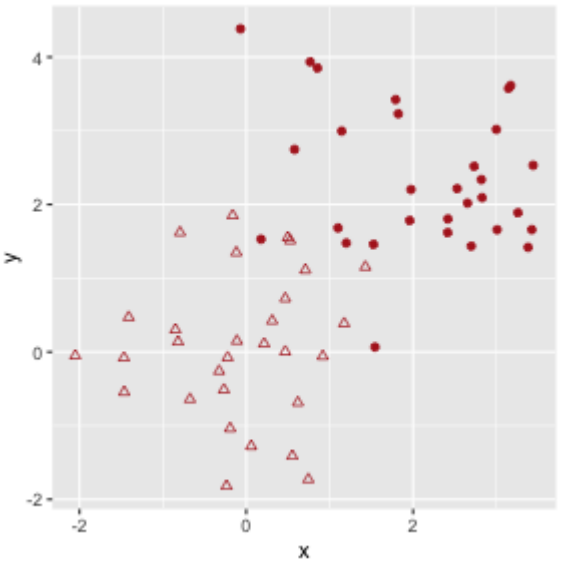


(Courtesy https://images.app.goo.gl/yF6R6XzVtPyoSBtb8)

K-Means Clustering: Example

Let's generate some fake data:

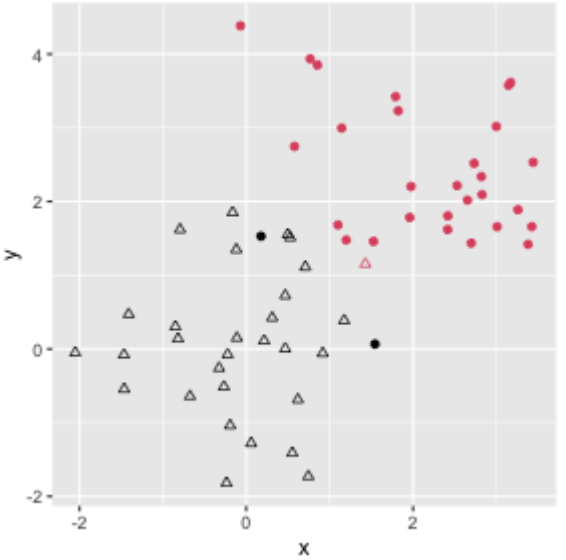
```
set.seed(101)
x <- c(rnorm(30),rnorm(30,mean=2.25))
y <- c(rnorm(30),rnorm(30,mean=2.25))
s <- c(rep(2,30),rep(19,30))
df <- data.frame(x,y)
library(ggplot2)
ggplot(data=df,mapping=aes(x=x,y=y)) +
  geom_point(color="firebrick",shape=s)
```



K-Means Clustering: Example

What happens if we assume two clusters?

```
km.out <- kmeans(scale(df),2,nstart=20) # run clustering in the standardized space...
color <- km.out$cluster
ggplot(data=df,mapping=aes(x=x,y=y)) + # ...but visualize in the native space
  geom_point(color=color,shape=s)
```



- Initially, the algorithm randomly associates data to clusters. To mitigate this aspect of randomness, set the `nstart` argument in the function call to a large number (e.g., 50); `R` will select the best result, the one with the smallest value of within-cluster variation.
- Note: we standardize the input to K-means, but visualize the results in the data's native space.

K-Means Clustering: Output

```
km.out
```

```
## K-means clustering with 2 clusters of sizes 31, 29
##
## Cluster means:
##            x          y
## 1 -0.7512305 -0.7453282
## 2  0.8030395  0.7967301
##
## Clustering vector:
##  [1] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 23.17382 23.00059
##  (between_SS / total_SS =  60.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

K-Means Clustering: Output

Look at the available components above:

totss: the average squared distance between any one data point and all other data points.

```
km.out$totss
```

```
## [1] 118
```

```
sum(dist(scale(df))^2)/length(x) # gets the same result!
```

```
## [1] 118
```

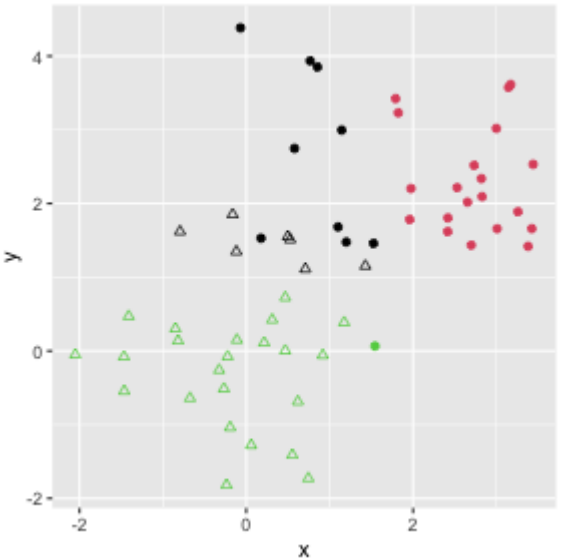
tot.withinss: the average squared distance between any one data point and all other data points *in its cluster*. We want this to be small relative to `totss`.

betweenss is `totss` - `tot.withinss`. We want this to be large relative to `totss`.

K-Means Clustering: Example

What happens if we assume three clusters?

```
km.out <- kmeans(scale(df),3,nstart=20)
color <- km.out$cluster
ggplot(data=df,mapping=aes(x=x,y=y)) +
  geom_point(color=color,shape=s)
```



It works (there is no reason why it wouldn't)...but the result visually appears less clean.

K-Means Clustering: Example

```
km.out
```

```
## K-means clustering with 3 clusters of sizes 16, 20, 24
##
## Cluster means:
##            x          y
## 1 -0.2969293  0.6183622
## 2  1.1850121  0.7282803
## 3 -0.7895572 -1.0191418
##
## Clustering vector:
##  [1] 3 3 3 3 3 3 3 3 3 1 1 1 3 3 3 3 3 3 1 1 3 3 3 3 3 1 3 1 2 2 2 1 2 1 2 2 1 2 2 2 1 2 1 2 2 3 2 1 1 2 2 1 2
##
## Within cluster sum of squares by cluster:
## [1] 10.84705  6.91868 14.12351
##  (between_SS / total_SS =  73.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter
## [9] "ifault"
```

As $k \to n$, `(between_SS/total_SS)` goes to 100%. 100% is not necessarily your goal: you'd be "overfitting" the data at that point. (Each point is its own "cluster.") Again: there is no universally agreed upon metric for "stopping" K-means as you increase k.

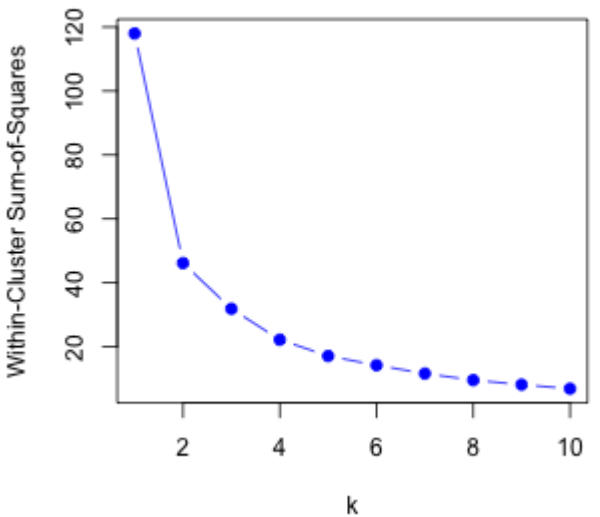
K-Means Clustering: Choosing k

However, just because there are no *universally agreed upon metrics* for choosing k doesn't mean there aren't any metrics at all. Some commonly used ones include

- the elbow method;
- the silhouette method; and
- the gap statistic.

K-Means Clustering: the Elbow Method

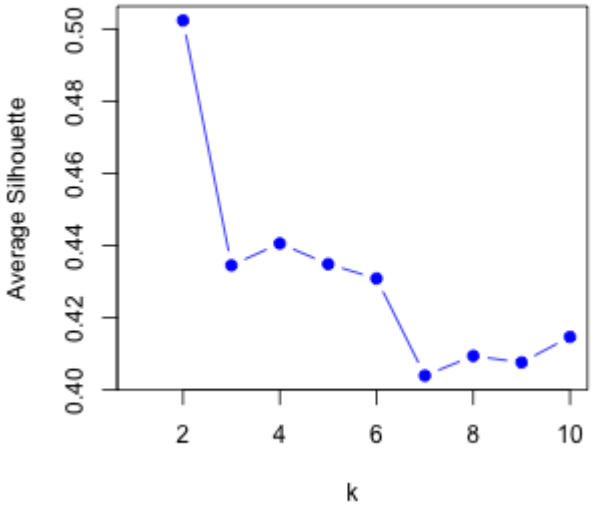
```
wss <- rep(NA,10)
for ( ii in 1:10 ) {
  km.out <- kmeans(scale(df),ii,nstart=20); wss[ii] <- km.out$tot.withinss;
}
plot(1:10,wss,xlab="k",ylab="Within-Cluster Sum-of-Squares",pch=19,col="blue",typ="b")
```



The "elbow" is around $k = 2$ (or 3?).

K-Means Clustering: the Silhouette Method

```
library(cluster)
ss <- rep(NA,10)
for ( ii in 2:10 ) {
  km.out <- kmeans(scale(df),ii,nstart=20); ss[ii] <- mean(silhouette(km.out$cluster,dist(scale(df)))[,3]);
}
plot(2:10,ss[2:10],xlab="k",ylab="Average Silhouette",pch=19,col="blue",typ="b",xlim=c(1,10))
```



The highest score is for k = 2, indicating that two clusters is optimal.

See, e.g., the wikipedia page on "Silhouette (clustering)" for details and the original citation.

K-Means Clustering: the Gap Statistic

The gap statistic method attempts, essentially, to perform on-the-fly hypothesis testing. (See here for more details.) Utilizing functions in the `cluster` and `factoextra` packages:

```
suppressMessages(library(factoextra))
gs <- clusGap(scale(df),FUN=kmeans,nstart=20,K.max=10,B=50)
```

```
## Clustering k = 1,2,..., K.max (= 10): .. done
## Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
## ................................................. 50
```

```
fviz_gap_stat(gs)
```