Unsupervised Learning I

36-600

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)
- "Unsupervised" simply means that none of the variables is a response variable: we are not trying to predict the value of a measurement
- 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
- A main, overriding issue with unsupervised learning is that *there are no universally accepted mechanisms for model assessment or selection, i.e., there is not necessarily going to be a unique right answer!*

Digression: Similarity

- Unsupervised learning relies on notions of *similarity*: how similar or dissimilar are two data?
- In the wider world of statistics and machine learning, there are many, many ways to quantify similarity
  - here we focus on the most intuitive, the L2-norm, better known as the Euclidean distance:

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

  - 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 assumes the Euclidean distance)
  - the output is a (symmetric) matrix
  - *beware*: if your data frame has more than about 20,000 rows, your computer may have insufficient memory to store the matrix

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 Euclidean 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*
  - if the ratio of the first number to the second number is small, we've found small tight clusters that lie far apart

Clustering

- What are the caveats?
  - there is *no guarantee* that the data are distributed across multiple effectively discontiguous clusters, i.e., unsupervised learning is never guaranteed to generate an "actionable result"
  - commonly applied clustering algorithms *are not applicable to categorical data* (however, we will mention some less commonly applied algorithms later)
  - in many clustering methods, *all* data are forced into clusters; this may not be optimal
  - any method that relies on distances will be impacted by units

Standardization

- Regarding "impacted by units"...
  - it is common practice to *standardize* (or *scale*) the data within each column of the data frame:

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

  - $\bar{X}$ and S_X are the sample mean and sample standard deviation of the data, respectively
  - the mean and the standard deviation of the scaled distribution will be 0 and 1, respectively
- To standardize the data separately in each column of a data frame, use the `scale()` function:

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

K-Means Clustering

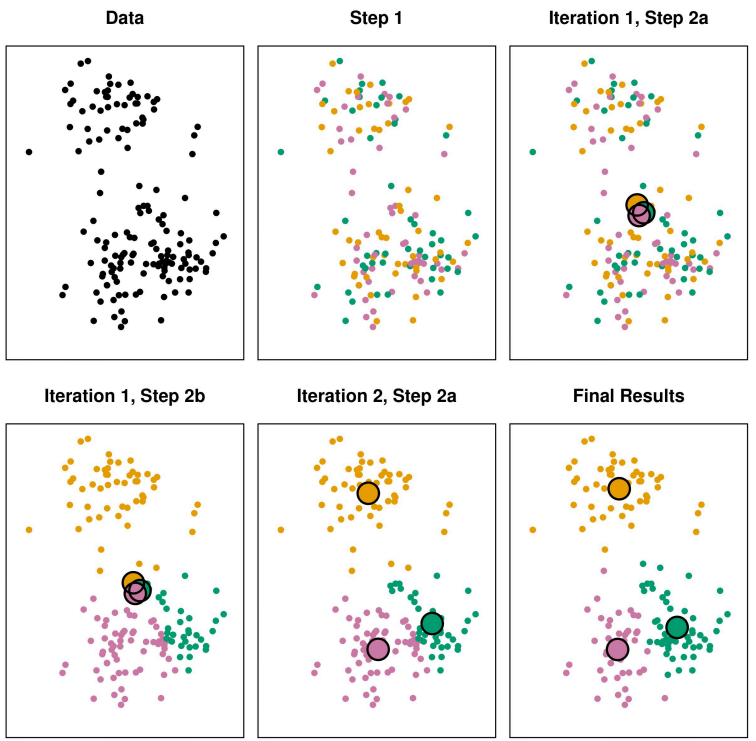
- The algorithm for K-means clustering is straightforward:

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).

- Note the following:
  - as previously stated, there is no universally accepted metric that would lead us to conclude that a particular value of K is the optimal one
  - your results can change from run to run unless you explicitly set a random number seed immediately before calling `kmeans()`

K-Means Clustering

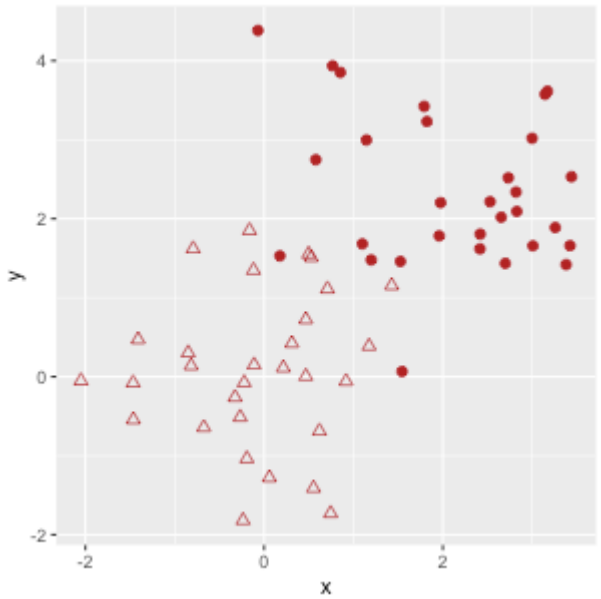


(Courtesy https://images.app.goo.gl/yF6R6XzVtPyoSBtb8)

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)
ggplot(data=df,mapping=aes(x=x,y=y)) +
  geom_point(color="firebrick",shape=s,cex=2)
```

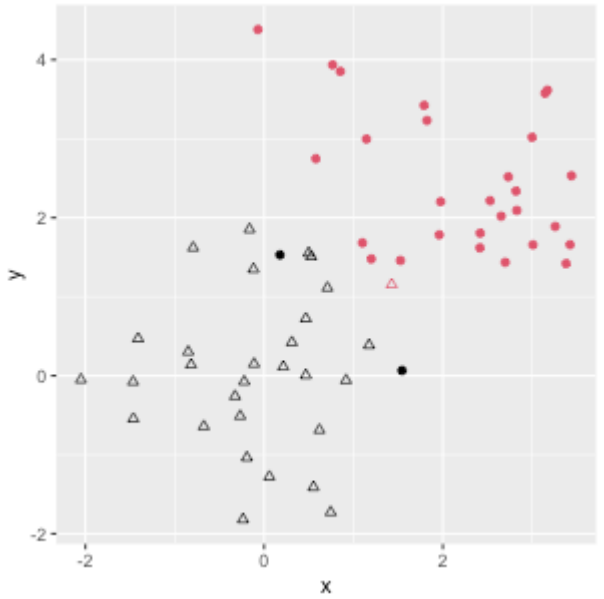


K-Means Clustering: Example

- What happens if we assume two clusters?

```
# run clustering in the standardized space...
km.out <- kmeans(scale(df),2,nstart=20)
color  <- km.out$cluster
# ...but visualize in the native space
ggplot(data=df,mapping=aes(x=x,y=y)) +
  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., 20)
- 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
## [39] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 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"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

K-Means Clustering: Output

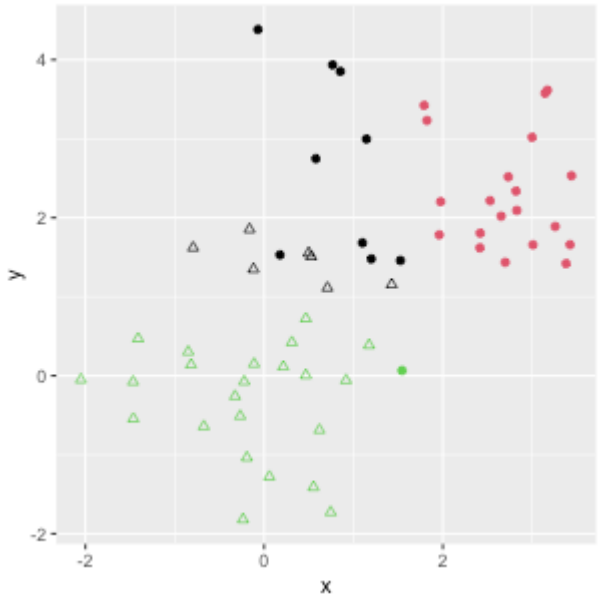
- Look at the output on the last slide:
  - *totss*: the average squared distance between any one data point and all other data points
  - *tot.withinss*: the average squared distance between any one data point and all other data points *in its cluster* (we want this to be small relative to `totss`)
  - *betweenss* is `totss - tot.withinss` (we want this to be large relative to `totss`)
- As $k \to n$, `(between_SS/total_SS)` goes to 100%
  - 100% is *not* the goal: you'd be "overfitting" the data at that point, with each data point in its own "cluster"

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 1 3 1 2 2 2 1 2 1 2 2
## [39] 1 2 2 2 1 2 1 2 2 3 2 1 1 2 2 1 2 1 2 2 2 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"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

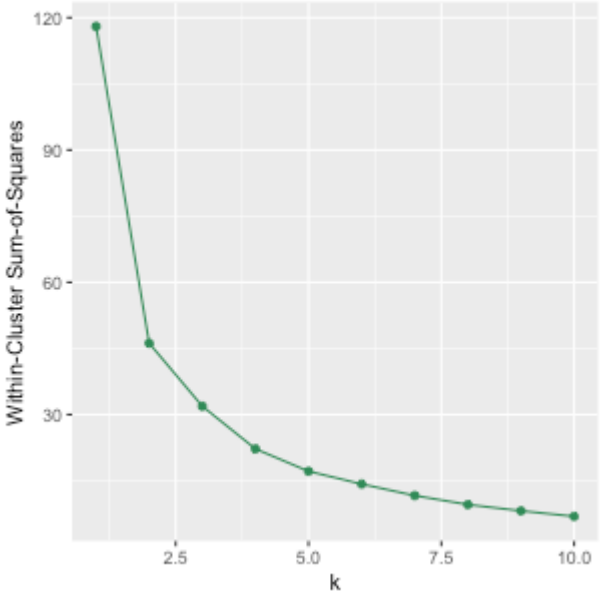
K-Means Clustering: Choosing k

- Just because there are no *universally agreed upon metrics* for choosing k doesn't mean there aren't any metrics at all...commonly used ones include:
  - the elbow method;
  - the silhouette method; and
  - the gap statistic

K-Means Clustering: the Elbow Method

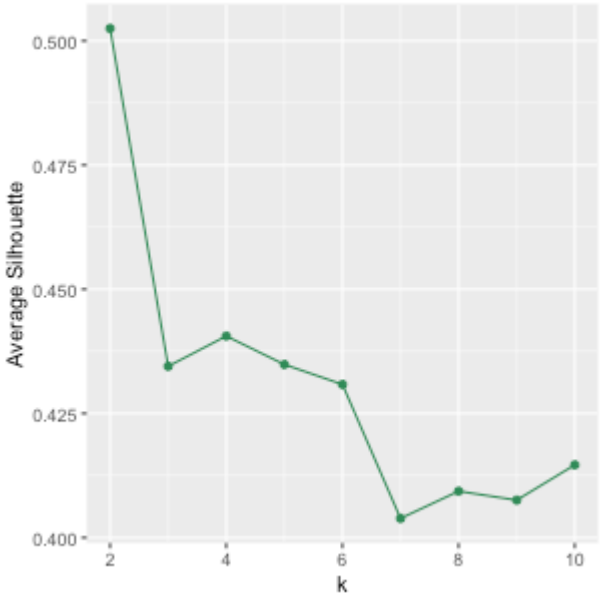
```
wss <- rep(NA,10)
for ( ii in 1:10 ) {
  km.out <- kmeans(scale(df),ii,nstart=20)
  wss[ii] <- km.out$tot.withinss;
}
df.plot <- data.frame("k"=1:10,wss)
ggplot(data=df.plot,mapping=aes(x=k,y=wss)) +
  geom_point(col="seagreen") +
  geom_line(col="seagreen") +
  ylab("Within-Cluster Sum-of-Squares")
```

- The "elbow" is around $k = 2$ (or 3?)
    - one should avoid using the elbow method



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])
}
df.plot <- data.frame("k"=2:10,"ss"=ss[2:10])
ggplot(data=df.plot,mapping=aes(x=k,y=ss)) +
  geom_point(col="seagreen") +
  geom_line(col="seagreen") +
  ylab("Average Silhouette")
```

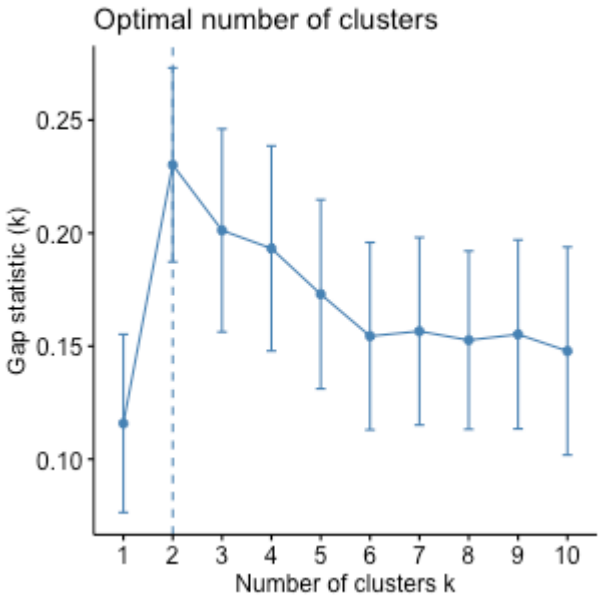


- The highest score is for k = 2, indicating that two clusters is optimal
- See, e.g., the wikipedia page on "Silhouette (clustering)"

K-Means Clustering: the Gap Statistic

- The gap statistic method attempts, essentially, to perform on-the-fly hypothesis testing

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



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

K-Prototypes and K-Modes

- What if our data have categorical variables?
- Two options for analyzing categorical or mix-type data are
    - *K-modes*: we would use this algorithm if our data consist *completely* of factor variables..an example implementation is `kmodes()` in `R`'s `klaR` package
    - *K-prototypes*: we would use this algorithm if our data consist of a mix of factor and numeric variables...an example implementation is `kproto()` in `R`'s `clustMixType` package
- Both of these algorithms alter the distance calculation to take into account the categorical nature of the factor variables
    - one can see more details about, e.g., K-prototypes in this paper