STAT 380 – Clustering Part 2 (Lecture 19)

NOTE: To perform a clustering analysis, the researcher has several decisions to make including:

- Will you screen for outliers? If found, what should you do with them?
- Which procedure should you use for clustering?
- Should you standardize the data?
- How many clusters should you use?
- How will you initialize the clusters?
- Since the clustering procedure may have an element of randomness (such as when using random initializations), we may find different results. How would you choose the best one?

K-means Clustering

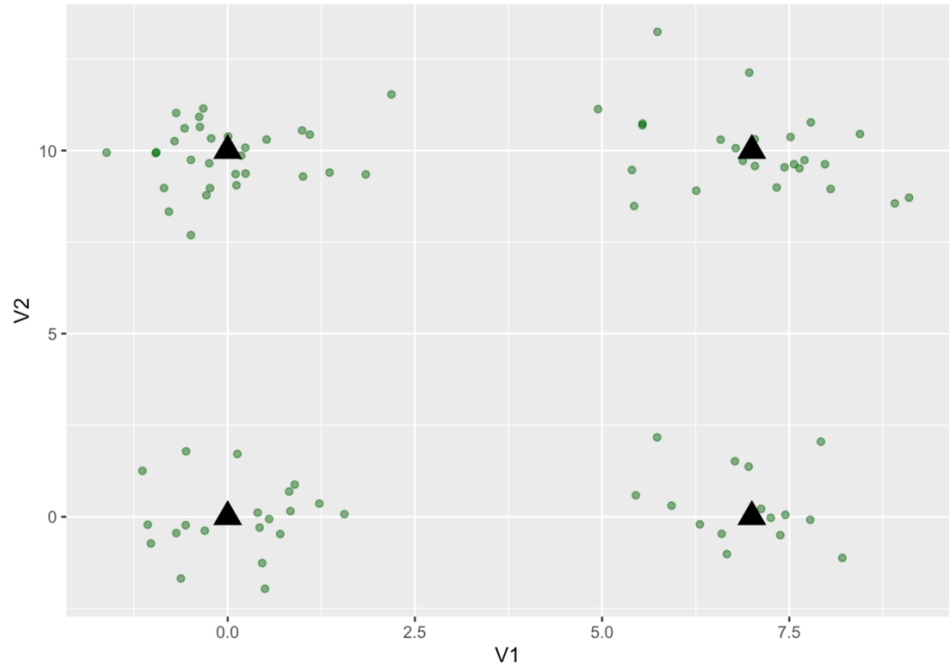
NOTE: One of the most simplistic, yet most commonly employed clustering methods is called K-means clustering. One variation of the K-means algorithm was presented in Lecture 18. This lecture illustrates one method for performing K-means clustering in R and provides a method for choosing the number of clusters.

K-means in R

EXAMPLE 1: Implement K-means clustering on a simulated dataset. Download STAT380_L19.Rmd.

a. Generate a simulated dataset using the provided code and plot the data. The true cluster centroids are shown as solid black triangles.

```
ggplot(data = df, mapping = aes(x = V1, y = V2)) +
  geom_point(alpha = 0.5, color = "darkgreen") +
  geom_point(data = data.frame(V1 = c(0, 7, 0 ,7), V2 = c(0, 0, 10, 10)),
             color = "black", size = 5, shape = 17)
```



b. Implement K-means clustering using the kmeans function from the stats package. (Since stats is part of base R, you do not need a special library command to access the kmeans function.) Specify that we want to use K=4 clusters and limit the number of iterations to 10. As we saw in the last lecture, different initializations can lead to different results. Specify that R uses 15 random initializations. Since the initializations will be determined randomly, specify a seed of 123.

```
set.seed(123)
kmeans_res4 <- kmeans(x = df, centers = 4, iter.max = 10, nstart = 15)
set.seed(NULL)
```

```
#View Results
kmeans_res4
```

```
## K-means clustering with 4 clusters of sizes 15, 25, 30, 20
##
## Cluster means:
##             V1          V2
## 1  6.901989039  0.3220471
## 2  7.024022701  9.9843161
## 3 -0.009798087  9.8624889
## 4  0.127085544 -0.0367189
##
## Clustering vector:
##  [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3
## [39] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2
## [77] 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 23.98346 59.62763 43.49340 31.16636
##  (between_SS / total_SS =  95.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

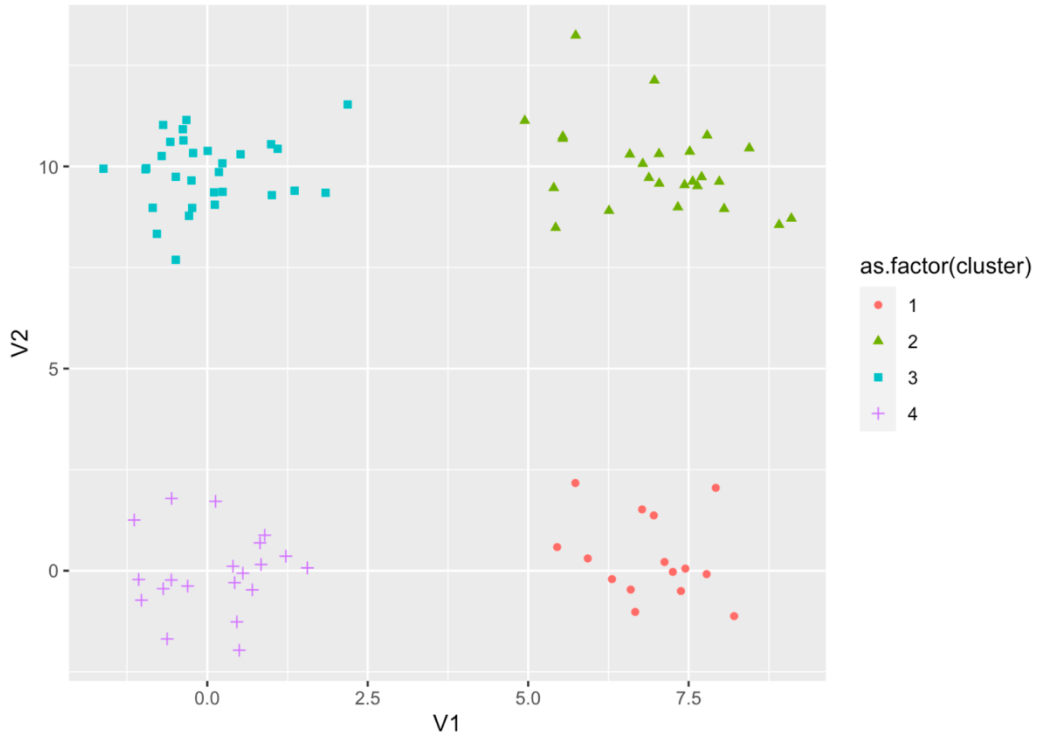
c. How many observations were assigned to each cluster? Plot the assignments.

```
#Find the number of observations assigned to each cluster
table(kmeans_res4$cluster)
```

```
##
##  1  2  3  4
## 15 25 30 20
```

```
#Plot cluster assignments
#First - add assignments to df
df2 <- data.frame(V1 = df$V1, V2 = df$V2, cluster = kmeans_res4$cluster)

#Visualize
ggplot(data = df2, mapping = aes(x = V1, y = V2,
                                 color = as.factor(cluster),
                                 shape = as.factor(cluster))) +
  geom_point()
```



d. Extract the within-sum of squares (WSS) for each component and the total across all components.

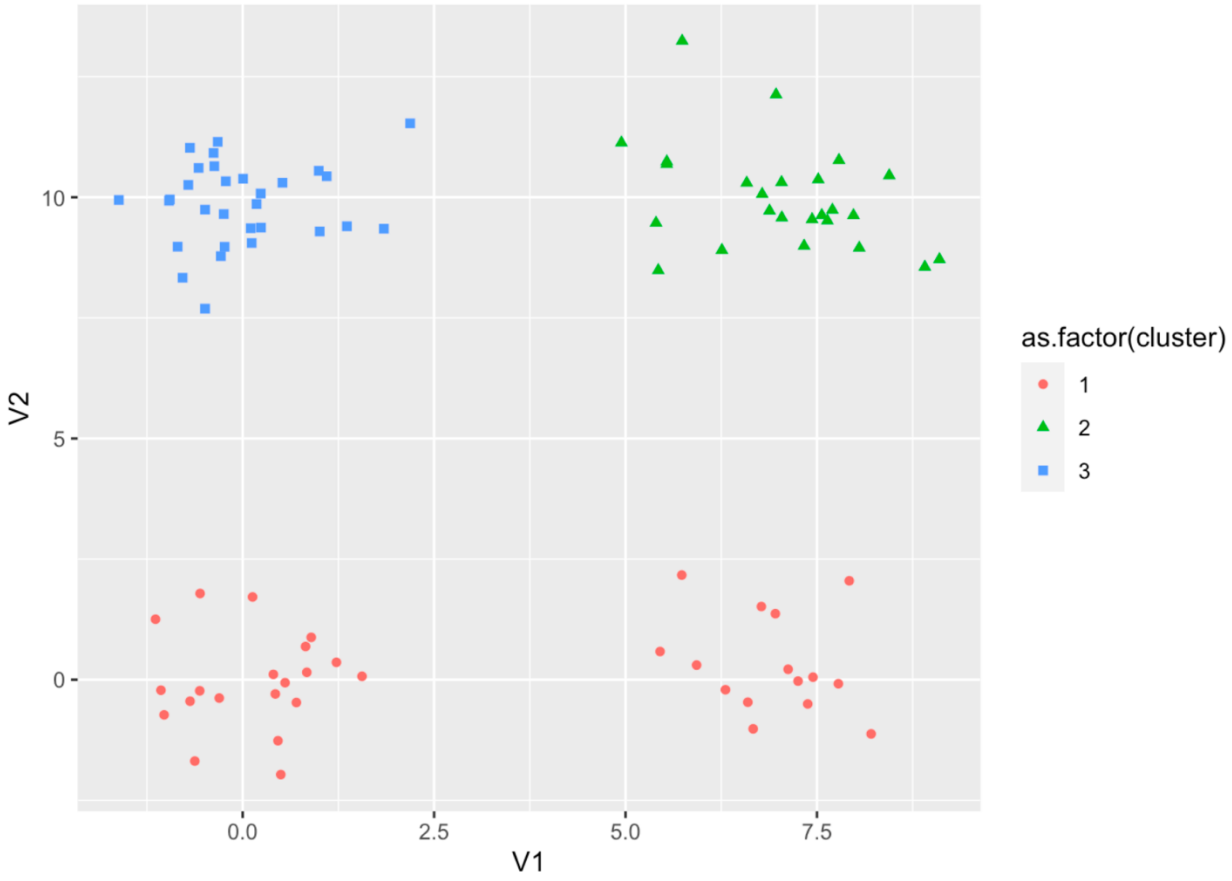
```
#Find WSS (for each component)
kmeans_res4$withinss
```

```
## [1] 23.98346 59.62763 43.49340 31.16636
```

```
#Find sum of WSS across all components
kmeans_res4$tot.withinss
```

```
## [1] 158.2709
```

EXAMPLE 2: Suppose we decided to use K=3 clusters instead of 4. Perform K-means using the same settings as before but with 3 clusters, plot the cluster assignments, and find the WSS summed across all components.




```
#Find WSS (for each component)
kmeans_res3$withinss
```

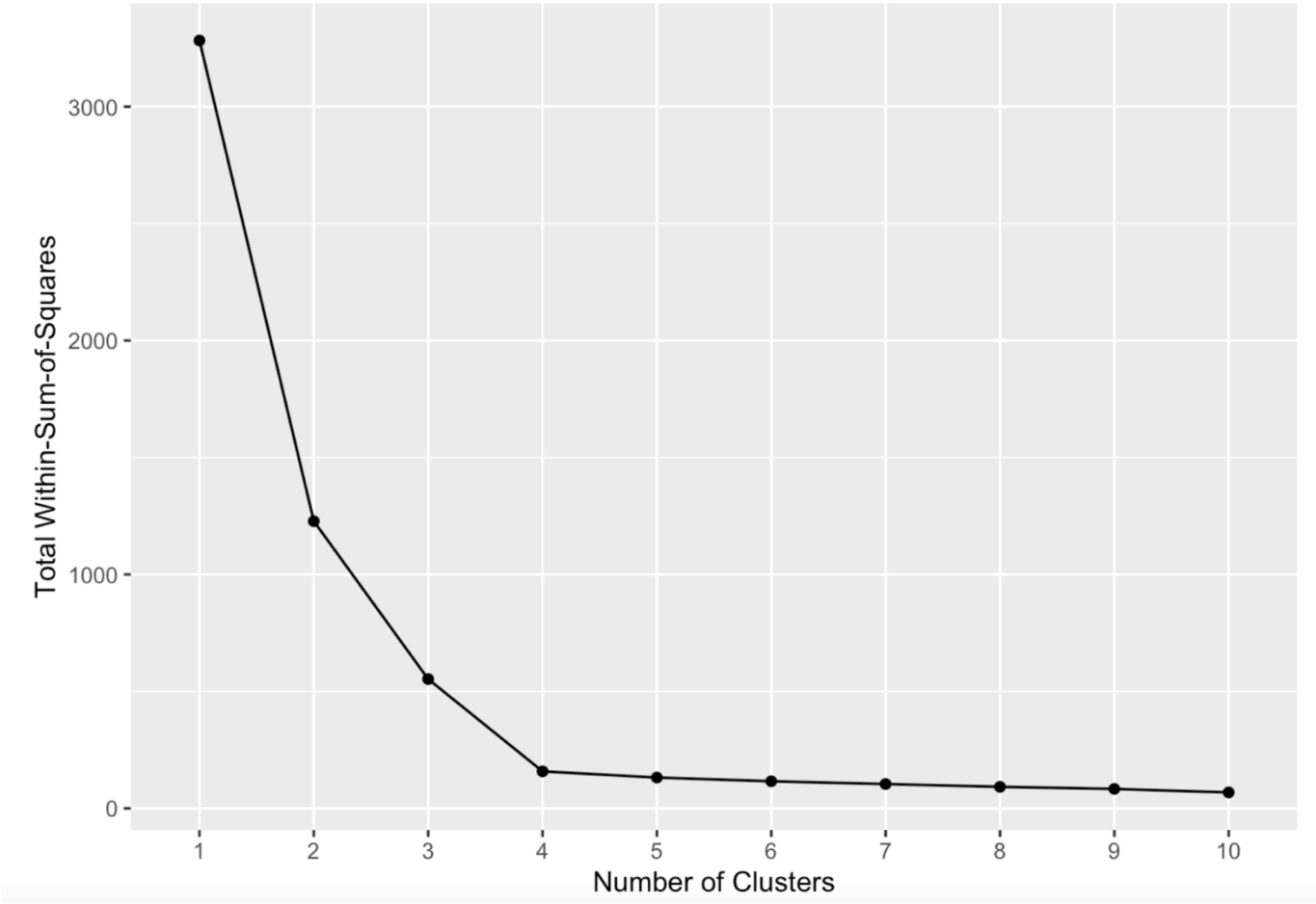
```
## [1] 449.67579  59.62763  43.49340
```

```
#Find sum of WSS across all components
kmeans_res3$tot.withinss
```

```
## [1] 552.7968
```

IDEA: In most instances, the number of clusters is unknown. We must determine the 'best' value of K, based on some criterion. One of the most common approaches for doing this is to use an elbow/scree plot of the WSS (total) for various values of K.

EXAMPLE 3: Using the dataset from Example 1 and 2, find the value of total WSS for values of K ranging from 1 to 10. Then create a plot showing the total WSS as a function of values of K. Which value of K is optimal?



NOTE: Since k-means is based on distance calculations, we generally standardize the X's before running the algorithm so that the scale of the inputs does not affect similarity/dissimilarity. To scale the data, you could use df <- scale(df, center = TRUE, scale = TRUE). In this simulated dataset, both V1 and V2 were on the same scale, so it was not as important to do so.

NOTE: K-means is one of the most commonly used clustering algorithms, but it is not without its disadvantages:

- Advantages
  - Easy to implement/understand
  - The algorithm is guaranteed to converge (but it is not guaranteed to find the globally optimal solution)
  - Many variations exist that make it adaptable
- Disadvantages
  - K must be specified
  - The basic form of K-means works well at finding spherical clusters, but other shapes may present a challenge
  - The results depend on the initialization (we solve this by using a number of different initializations)
  - K means struggles when finding of varying sizes and densities
  - Outliers can present a problem
  - K-means is a "hard" clustering procedure (i.e., assigns each observation to a cluster rather than an assigning a degree of membership)