Final Project – K-means

Nolan McCafferty

Introduction

I have chosen to explore the k-means clustering algorithm for my final project. Clustering in general is a set of techniques for finding groups of observations with a data set. Thus, the goal is to have observations in the same group be similar to each other and observations in different groups to be different. Clustering is an unsupervised method because we do not have a response variable to train on. K-means is the simplest and most commonly used clustering algorithm for splitting a data set into k groups. The reason I have chosen to analyze K-means is because I was once asked about the specifics of it in an ineterview and did not know the answer (Yikes).

Set Up

The first step in classifying observations into groups is to determine the metric used for "similarity". To evaluate the similarity of observations we use a distance measurement. The most common measure is *Euclidean* distance:

$$d_{euc}(x1, x2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2} \tag{1}$$

where x_1 and x_2 are n-length vectors.

Other methods include *Manhattan* distance, which uses the absolute value, and various correlation-based distance measures. The Pearson correlation distance is defined as follows:

$$d_{pear}(x_1, x_2) = 1 - \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})(x_{2i} - \bar{x_2})}{\sqrt{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})^2 \sum_{i=1}^{n}(x_{2i} - \bar{x_2})^2}} \tag{2}$$

The k-means clustering algorithm partitions a data set into k groups (clusters). The number of clusters k is pre-specified by the analyst. For successful clustering, we want high intra-cluster similarty and low inter-cluster similarity. Each of the clusters is represented by its centroid–the mean of the points assigned to the cluster. The key idea behind the k-means algorithm is to minimize the total intra-cluster variation. While there are several k-means algorithms out there, we will focus on the standard one: the Hartigan-Wong algorithm (1979). The HW algorithm defines the total intra-cluster variation as the sum of squared Euclidean distances between observations and their respective centroids:

$$Z(c_j) = \sum_{x_i \in c_j}(x_i - \mu_j)^2 \tag{3}$$

where x_i is an observation in cluster c_j and μ_j is the mean of the observations in cluster c_j. Thus, the goal is to minimize this value over all the clusters:

$$min \sum_{j=1}^{k} Z(c_j) = min \sum_{j=1}^{k} \sum_{x_i \in c_j}(x_i - \mu_j)^2 \tag{4}$$

Algorithm

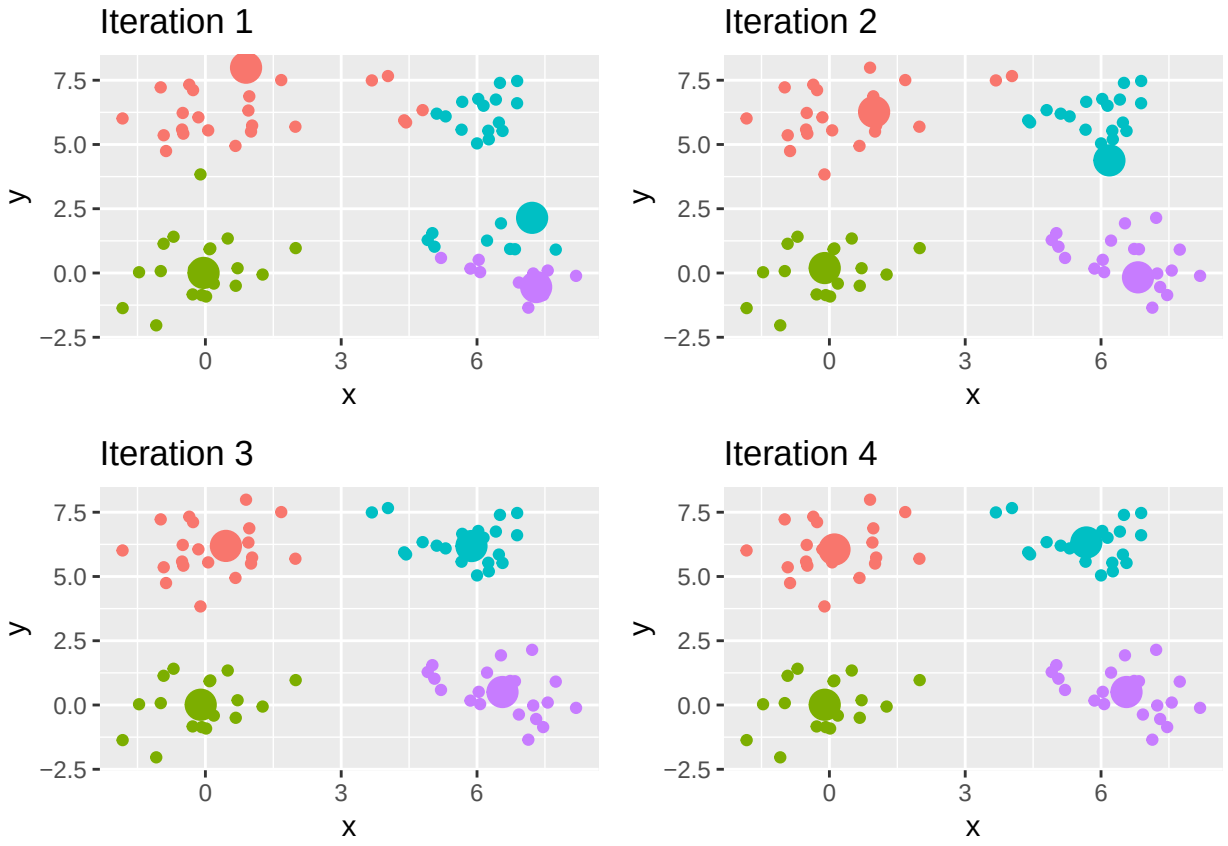
1. Choose the number of clusters k that we want the observations grouped into.
2. Select k random observations from the data to serve as the initial centroids for the clusters.
3. Assign each observation to their closest centroid using Euclidean distance.
4. For each cluster, update the centroid by calculating the new mean values of all the observations in the cluster. The centroid for the jth cluster is a p-dimensional vector containing the means of all the variables for the points in the jth cluster–where p is the number of variables.
5. Minimize the total intra-cluster sum of squares (Eq. 4), i.e. iterate steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached (10 by default in R).

Here is my code for a function that executes k-means clustering:

```
# function takes the data as a data frame and the number of clusters
my.k.means <- function(data, num.clusters) {
  num.iter <- 4
  clusters <- list()
  centroids <- list()
  for (j in 1:4) {
    if (j == 1) {
      centroids[[j]] <- data %>%
        sample_n(num.clusters)
    } else {
        centroids[[j]] <- setNames(data.frame(matrix(ncol=2, nrow=num.clusters)),
                                   c("x", "y"))
        for (m in 1:num.clusters) {
          centroids[[j]][m,] <- colMeans(data[which(clusters[[j-1]] == m),])
        }
    }
    cluster <- c()
    for (i in 1:nrow(data)) {
      dist <- c()
      for (k in 1:num.clusters) {
        dist[k] <- sum((data[i,] - centroids[[j]][k,])^2)
      }
      cluster[i] <- which.min(dist)
    }
    clusters[[j]] <- cluster
  }
  return(list(labels=clusters, means=centroids))
}
```

Simple Example

The following is a simple 2D example of my k-means function. The plots show the evolution of the clustering through the four iterations of the algorithm.

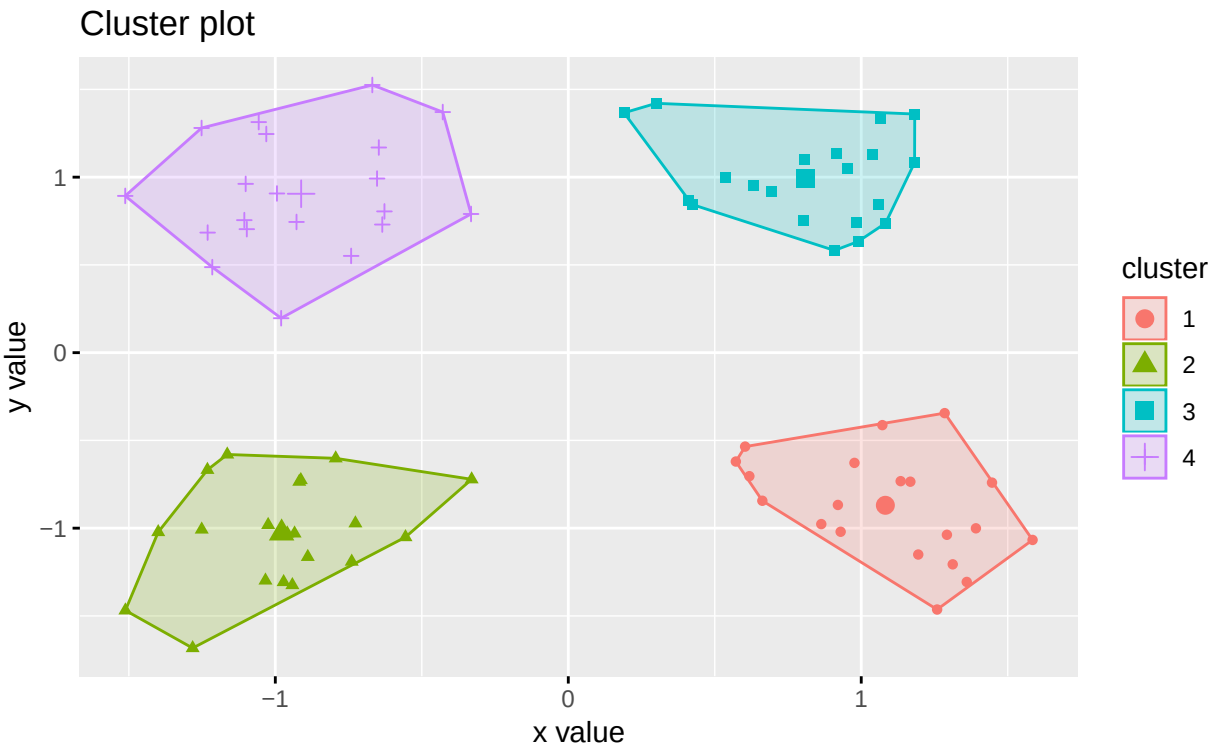


R Function

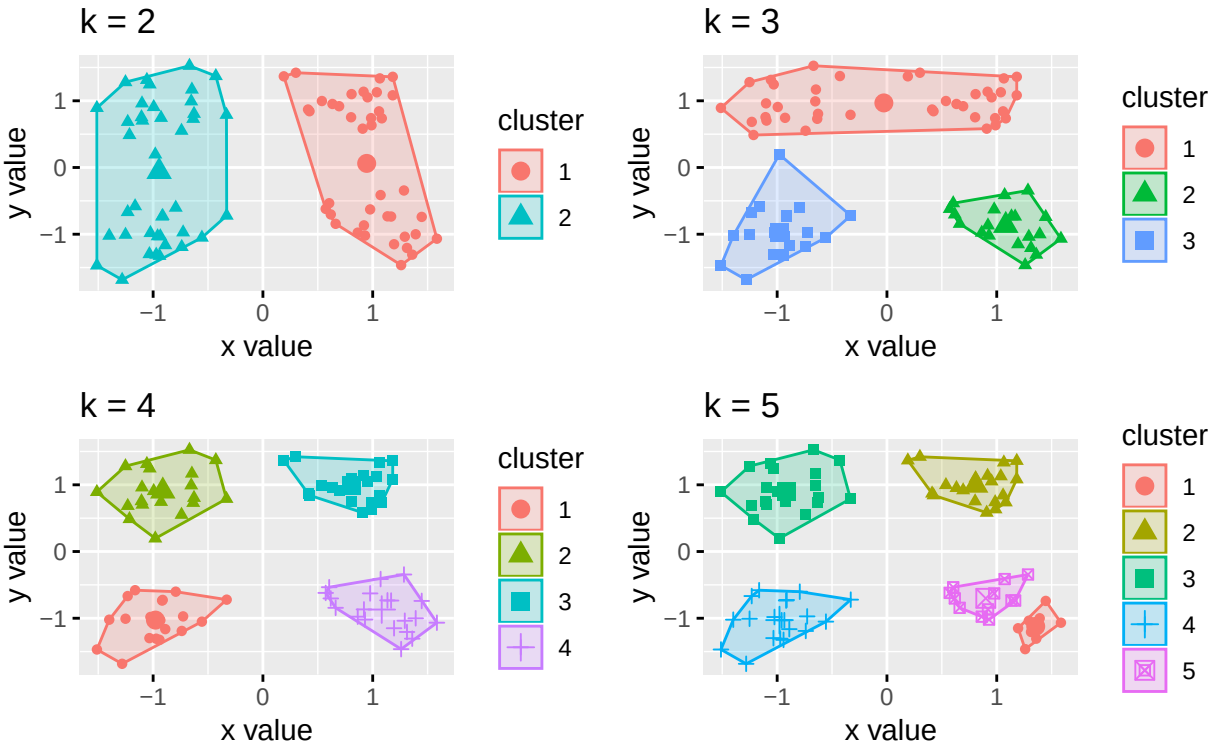
As usual, R has a convenient built in function called `kmeans`. The `kmeans` function takes *centers*, *max.iter*, *nstart*, and *algorithm* as parameters. Unsurprisingly, *max.iter* is the maximum number of iterations for the algorithm to run through. The parameter *centers* can either be a list of the coordinates for the centroids or k–the number of centroids. If *centers* is a number, *nstart* is the number of initial configurations to run through, picking the iteration that produces the lowest final intra-cluster variance. It is often recommended to set $nstart = 25$ to generate 25 initial configurations. Finally, *algorithm* gives a choice of the k-means algorithm to use (Hartigan-Wong, Lloyd, Forgy, or MacQueen) and used our HW algorithm by default. The output of the `kmeans` function includes a vector of integers indicating the cluster to which each observation is designated, the cluster centers, the number of points in each cluster, each intra-cluster sum of squares, and the total intra-cluster sum of squares.

There is also a handy function in the R package `factoextra` called `fviz_cluster` to visualize the output from `kmeans` nicely. If the observations have more than 2 dimensions, `fviz_cluster` will plot the first two principle components that explain the majority of the variance after performing principle component analysis (PCA). Here is what looks like for our toy data set:

```
clustered.data <- kmeans(data, centers = 4, nstart = 25)
fviz_cluster(clustered.data, data = data, geom = "point")
```

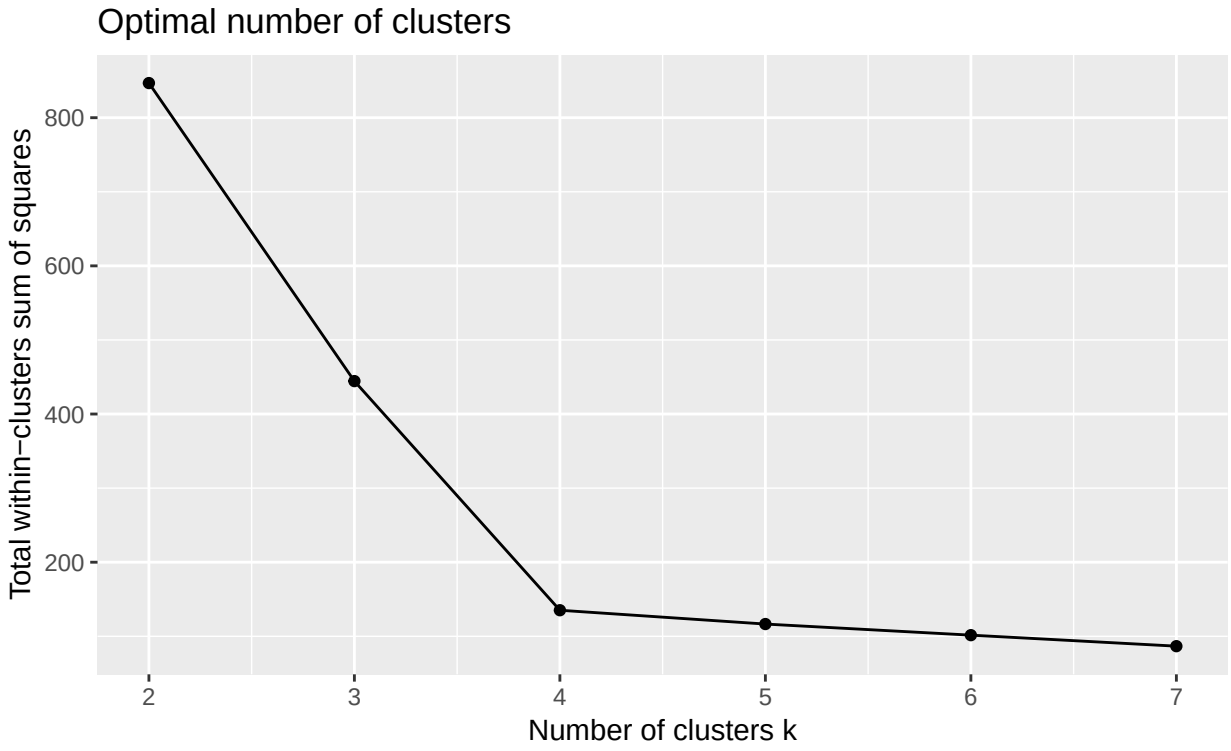


Since the number of clusters k must be given to the algorithm, it is important that the analyst picks the best value of k. One way to ensure this is to try out several values of k and pick the clustering that gives the lowest total intra-cluster sum of squares. We can try this out with our simple data set, even though we know the correct clustering should be with $k = 4$:



The plots above give a visual representation of the clusterings with a differing number of clusters, but to find

the optimal k we can use the Elbow method. In the Elbow method we compute the total intra-cluster sum of squares for each clustering and plot our results. Then, we look for the "bend in the knee" to tell us the optimal number of clusters. In R, this can be done using the function `fviz_nbclust`:

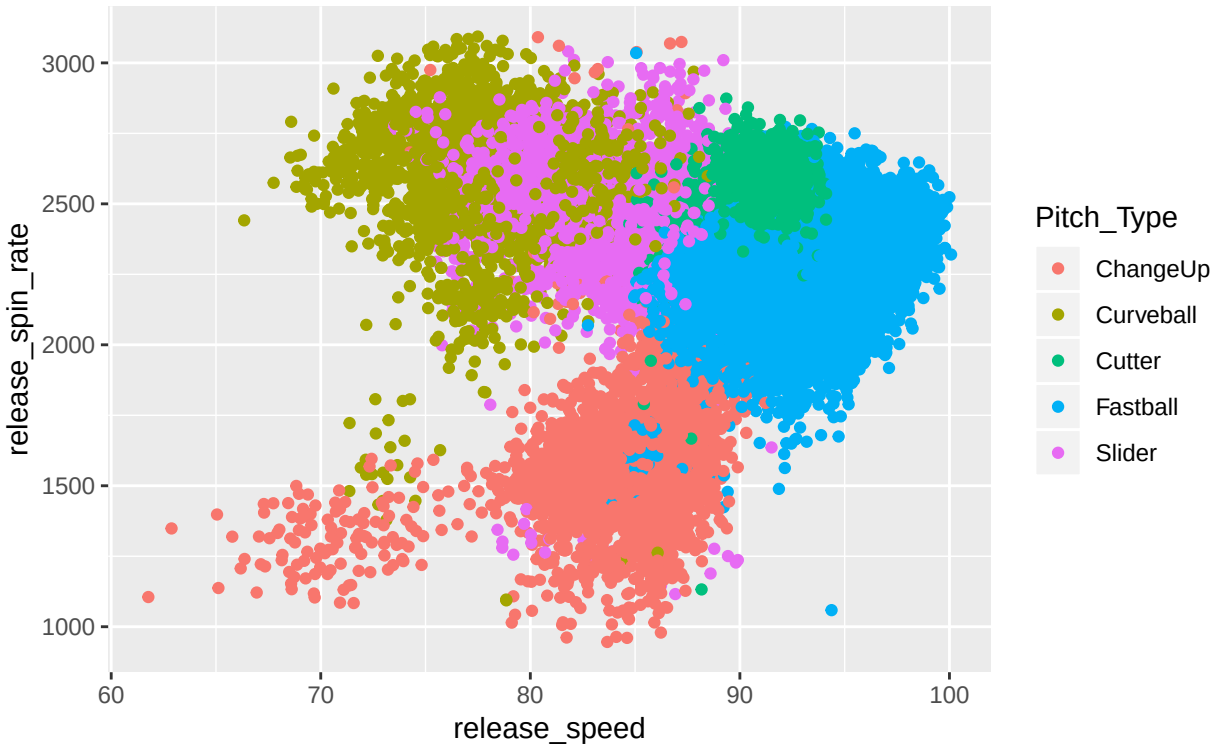


As you can see, this method correctly identifies the optimal number of clusters $k = 4$.

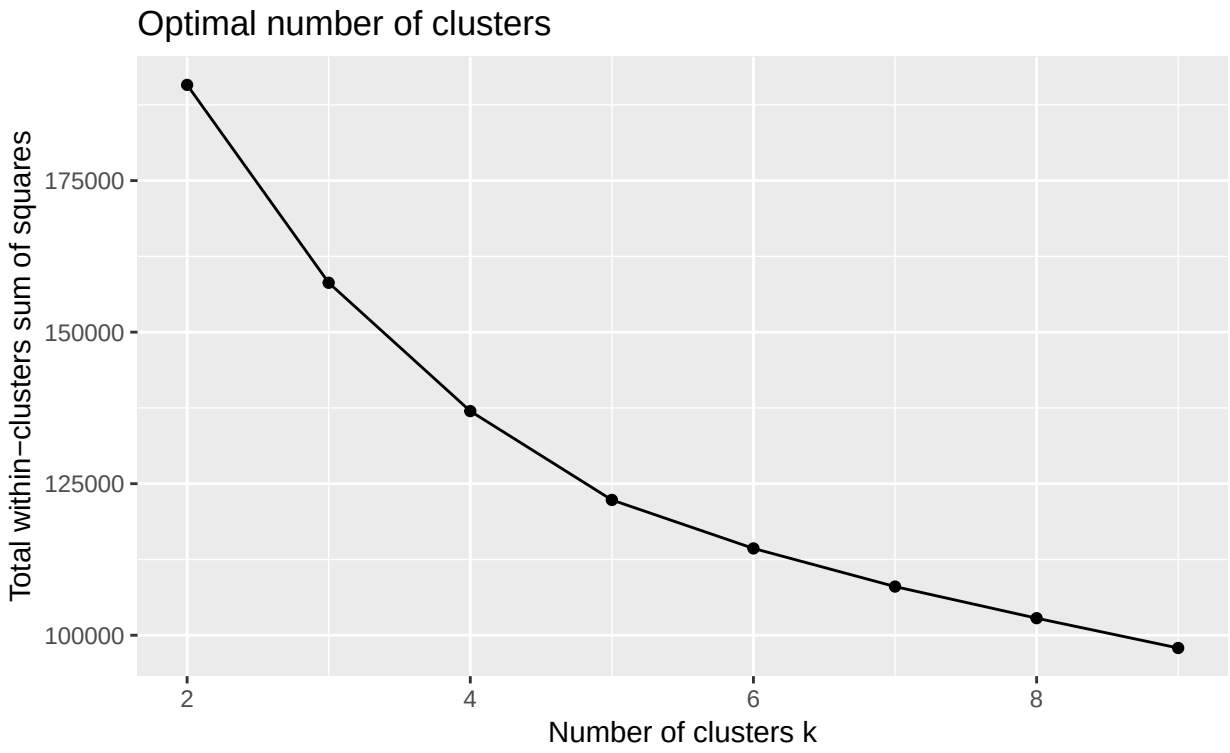
Slightly More Interesting Example

Now that we understand the basics of the k-means algorithm, we can utilize it on a much more interesting data set. The data that I am going to use was given to me as part of the interview process for the Atlanta Braves. The data set is Trackman pitch-level data from the 2018 MLB season. The goal is to see if we can cluster pitches together by their pitch type using k-means. The pitch-types in this data set are Fastball, Cutter, Slider, Curveball, and Changeup. Even though in this case we know that the optimal number of clusters should be five, we can continue with this analysis assuming we did not know the specific pitch types.

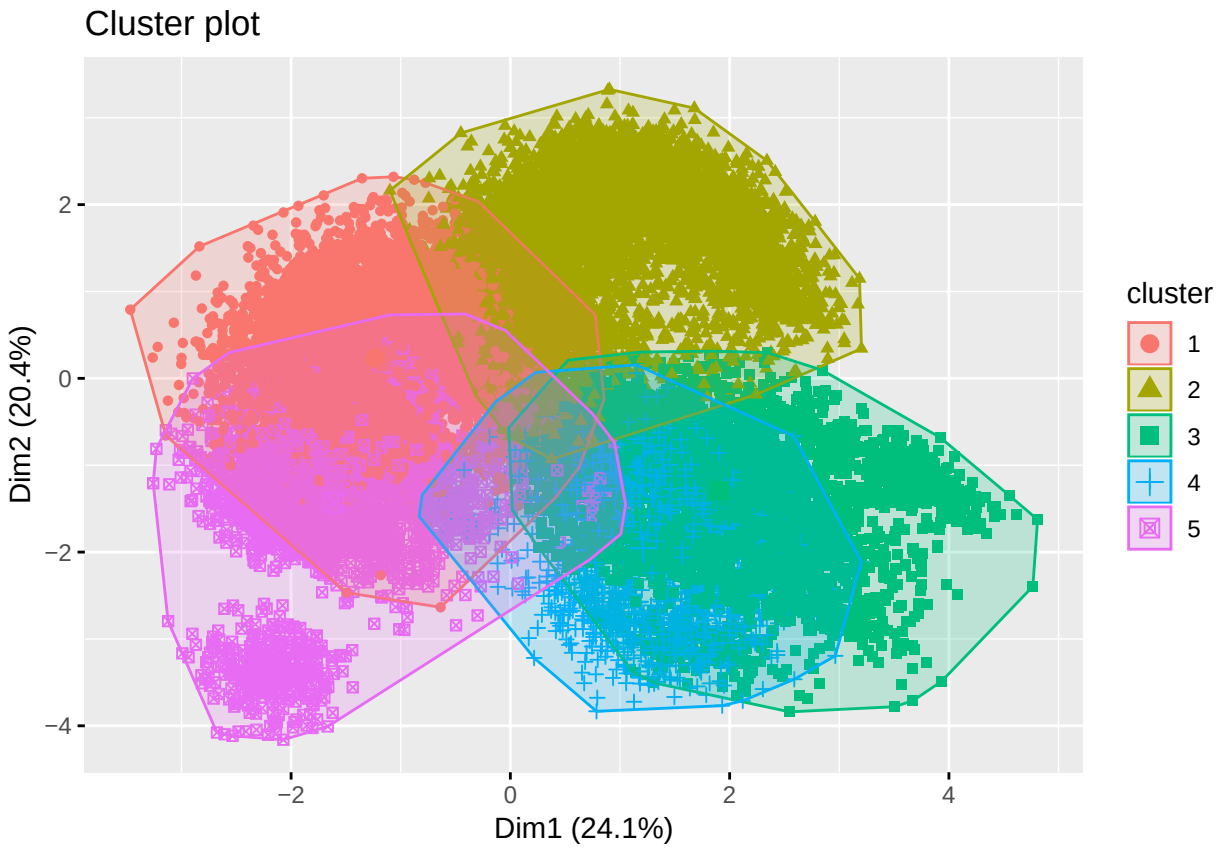
An important thing to note about the k-means algorithm is that the variables need to be scaled properly for the best results. We did not have to do this in our previous example because we used data that all had standard deviation 1. The variables that we will use to cluster pitches are: release speed, x-movement, z-movement, release spin rate, spin direction, release position x, release position z, release extension, plate x, and plate z. For reference, here is plot of the release speed and spin rate by pitch type:



Now we will implement k-means. Below we can see the elbow plot of the clustering:



We can see that the optimal number of clusters seems to be four or five. This makes sense, since cutters are very similar to fastballs and sliders and can be easily classified as either one without knowing who the pitcher is. If we knew who the pitcher was, we could much more easily classify those pitches as cutters. Using $k = 5$, we can examine our clusters and their first two principle components:



We can see that the first principle component explains 24.1% and the second only 20.4%. This is unsurprising given the dimensionality of the pitch data. Analyzing these clusters even more we can figure out which cluster belongs to which pitch:

Cluster	release_speed	release_spin_rate	x_movement	z_movement	spin_dir
1	93.16636	2293.207	-5.236643	7.7472931	210.46260
2	91.98110	2141.505	4.334484	7.5635636	153.42891
3	82.34081	2499.769	3.022902	-0.8492566	86.97067
4	79.20460	2618.928	-4.337941	-6.2116051	314.97325
5	85.60597	1793.377	-8.071137	2.7009159	253.30450

Pitch_Type	release_speed	release_spin_rate	x_movement	z_movement	spin_dir
ChangeUp	84.26710	1607.739	-2.9110821	4.1123505	204.9460
Curveball	78.14968	2586.148	0.6706065	-5.7828388	164.9980
Cutter	90.29020	2439.356	0.3168102	5.5172791	179.1913
Fastball	93.34690	2241.125	-2.5842469	7.7611604	196.5299
Slider	84.32108	2450.931	1.1291768	0.5411807	143.7359

Although the pitches are not perfectly clustered, we can see that cluster 1 represents curveballs, cluster 2 represents fastballs, cluster 3 sliders, cluster 4 cutters, and cluster 5 is changeups.

Conclusion

K-means clustering is a standard unsupervised learning algorithm that most every analyst should be familiar with (especially if you want to get a job with the Tigers I guess). I have presented the simplest variation of k-means–there are many more specific extensions out there. The algorithm is an efficient way to cluster data with numerical variables to get a better understanding of the data you are working with. Plus, it comes with cool visualization tools to enhance any explanation or argument.