# K Means Algorithm

*Chris Dong*

```
library(tidyverse)
library(stringr)
library(magrittr)
library(pdist)
library(microbenchmark)
```

```
rm(list=ls()) #remove variables in environment
gc() #garbage collection
```

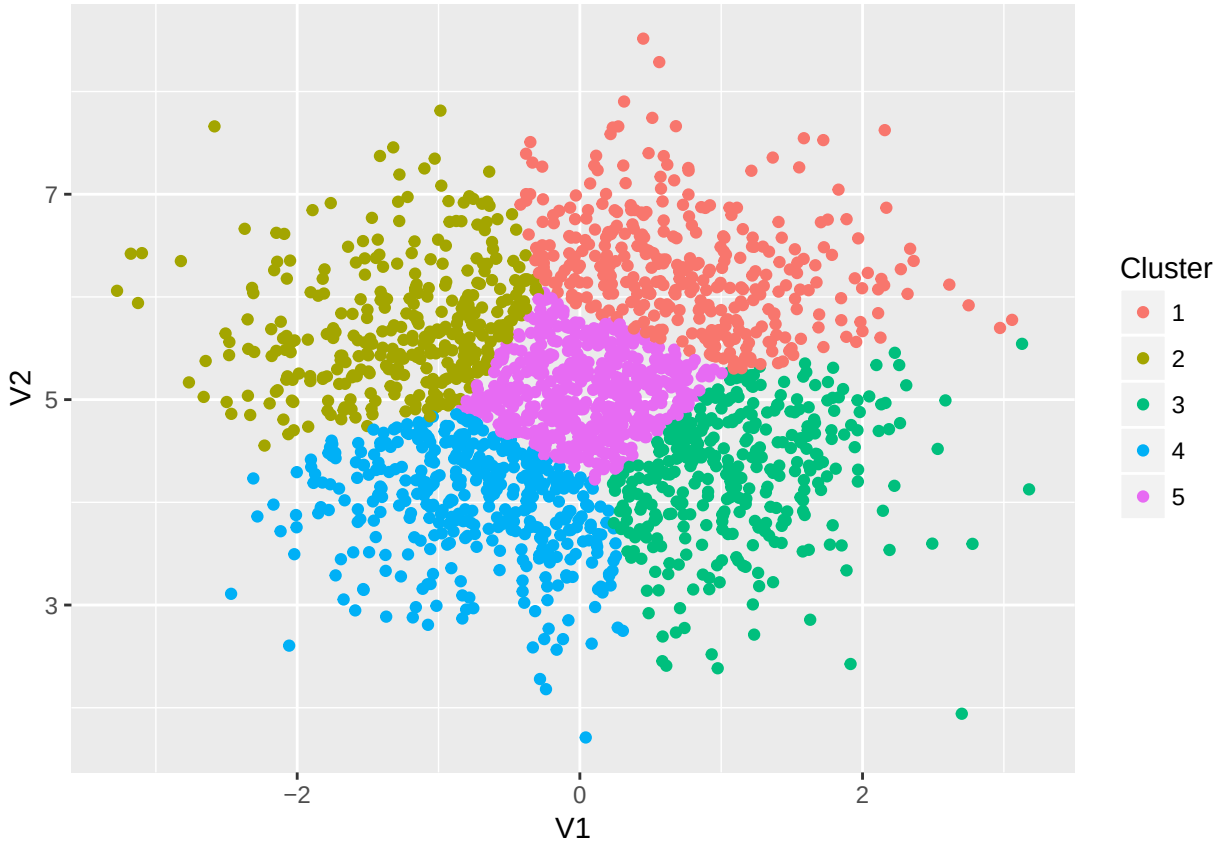
```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  728967 39.0    1168576 62.5   940480 50.3
## Vcells 1180074  9.1    2060183 15.8  1649043 12.6
```

```
set.seed(101)
myDF <-as_data_frame(matrix(rnorm(10000), ncol = 5))
myDF[2] <- myDF[2] + 5
myDF[3] <- myDF[3] - 0.1
myDF[4] <- myDF[4] + 1
myDF[5] <- myDF[5] - 3

kMeanAlg <- function(nReps = 10, myScatterInput = myDF, myClusterNum = 5, maxIter = 10000){
  mean <- function(x) sum(x)/length(x) #slightly faster than normal mean()
  #saving results
  mindiff <- vector("list", nReps)
  bestresult <- vector("list", nReps)
  bestcluster <- vector("list", nReps)


  #repeat nReps times
  for(times in seq_len(nReps)){
    #randomly assign points to each cluster
    cluster <- sample(seq_len(myClusterNum), size = nrow(myScatterInput),  replace = T)
    numiter <- 0
    while(numiter < maxIter){

      numiter <- numiter + 1 #keep count of number of iterations
      #compute cluster centroid
      center <- sapply(myScatterInput, function(x) tapply(x, cluster, mean))

      #compute euclidean distance from centroid
      diff <- as.matrix(pdist::pdist(myScatterInput, center))

      newcluster <- max.col(-diff, "first") #identify minimum and set new cluster

      if(identical(cluster, newcluster)) break # if cluster assignment unchanged, break
      cluster <- newcluster
    }

    #calculate sum of difference from center
```

```
    mindiff[[times]] <- sum(vapply(seq_along(diff[,1]),
                        function(i) diff[i, cluster[i]], numeric(1)))
    bestcluster[[times]] <- cluster
  }

  index <- max.col(-unlist(mindiff, F, F), "first")[1] #identify best result

  if(ncol(myScatterInput)==2){ # if 2-D
    a <- ggplot(myScatterInput, aes(x=V1,y=V2,color=factor(bestcluster[[times]]))) +
      geom_point() + labs(colour = "Cluster")
    print(a)
  } else if(ncol(myScatterInput)==3){ # if 3-D
    with(myScatterInput,scatterplot3d::scatterplot3d(x = V1, y = V2, z = V3,
                         color = factor(bestcluster[[times]])))
  }
return(unlist(mindiff[index], F, F))
}

paste("The sum of the Euclidean distances from their respective centroids is",kMeanAlg())
```

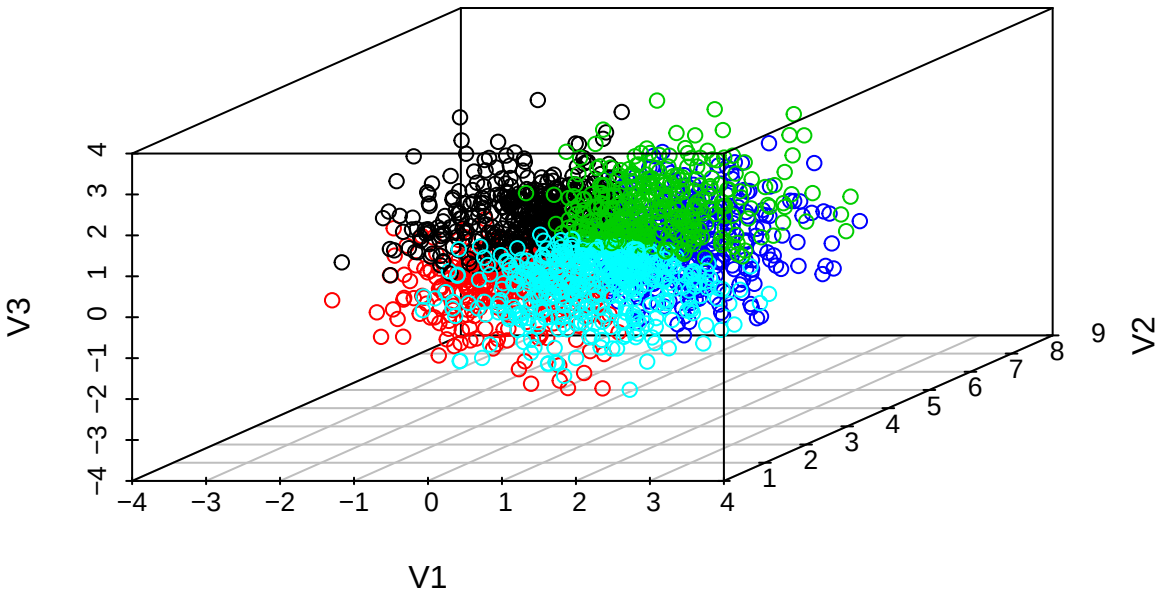
```
## [1] "The sum of the Euclidean distances from their respective centroids is 3375.33261068165"
```

```
kMeanAlg(myScatterInput = myDF[,1:2])
```




```
## [1] 1364.085
```

```
kMeanAlg(myScatterInput = myDF[,1:3])
```




```
## [1] 2287.233
```

```
kMeanAlg(myClusterNum = 10)
```

```
## [1] 2941.422
```

```
microbenchmark::microbenchmark(kMeanAlg(nReps = 1), kmeans(myDF, centers = 5, iter.max = 10000))
```

```
## Unit: milliseconds
##                                         expr       min        lq      mean
##                           kMeanAlg(nReps = 1) 28.518329 54.558966 88.769889
##  kmeans(myDF, centers = 5, iter.max = 10000)  2.090889  3.148366  4.750051
##     median        uq       max neval
##  74.119365 120.40100 222.16054   100
##   4.023633   5.00332  56.75454   100
```

```
microbenchmark::microbenchmark(kMeanAlg())
```

```
## Unit: milliseconds
##        expr      min       lq     mean   median       uq      max neval
##  kMeanAlg() 539.6635 759.666 840.6843 829.3884 922.3439 1189.938   100
```

After spending about 40 hours, my algorithm is about 20 times slower than the default `kmeans` function. Initially, my algorithm was about 10 minutes long and I nitpicked every single function to see if there is a faster way of doing the same thing.

**Noticable differences:**
`pdist` package is MUCH faster for computing euclidean distances.
`max.col(-diff, "first")` is quite a bit faster than `apply(diff, 1, which.min)`.
`seq_len()` is slightly faster than `1:n`
Preallocating the size of my `list` seems to improve the speed slightly as well.
`sum(x)/length(x)` is slightly faster than `mean(x)`, I think..Also, I found that putting my user-defined mean function inside was faster than outside my kmeans function. Not sure if it's a local vs global issue.
`vapply` is similar to sapply or lapply except that you specify the `class` and `length` of the output of `class`.
`unlist` by default uses `recursive = T` and `use.names = T` and is a little faster when we set it to false since keeping track of names isn't needed.
**Useful link:**
https://www.r-bloggers.com/faster-higher-stonger-a-guide-to-speeding-up-r-code-for-busy-people/