K-Means Clustering

Manuel Amunategui

June 13, 2016

Let's use our pipeline to prepare a data set for a k-means (https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html) unsupervised model.

First we need to load into memory the following functions:

Impute_Features

```
Impute_Features <- function(data_set, features_to_ignore=c(),
                            use_mean_instead_of_0=TRUE,
                            mark_NAs=FALSE,
                            remove_zero_variance=FALSE) {
     for (feature_name in setdiff(names(data_set), features_to_ignore)) {
          print(feature_name)
          # remove any fields with zero variance
          if (remove_zero_variance) {
               if (length(unique(data_set[, feature_name]))==1) {
                    data_set[, feature_name] <- NULL
                    next
               }
          }
          if (mark_NAs) {
               # note each field that contains missing or bad data
               if (any(is.na(data_set[,feature_name]))) {
                    # create binary column before imputing
                    newName <- paste0(feature_name, '_NA')
                    data_set[,newName] <- as.integer(ifelse(is.na(data_set[,feature_name]),1,0)) }
               if (any(is.infinite(data_set[,feature_name]))) {
                    newName <- paste0(feature_name, '_inf')
                    data_set[,newName] <- as.integer(ifelse(is.infinite(data_set[,feature_name]),1,0)) }
          }
          if (use_mean_instead_of_0) {
               data_set[is.infinite(data_set[,feature_name]),feature_name] <- NA
               data_set[is.na(data_set[,feature_name]),feature_name] <- mean(data_set[,feature_name], na.rm=TRU
E)
          } else {
               data_set[is.na(data_set[,feature_name]),feature_name] <- 0
               data_set[is.infinite(data_set[,feature_name]),feature_name] <- 0
          }
     }
     return(data_set)
}
```

Get_Free_Text_Measures

```
Get_Free_Text_Measures <- function(data_set, minimum_unique_threshold=0.9, features_to_ignore=c()) {
     # look for text entries that are mostly unique
     text_features <- c(names(data_set[sapply(data_set, is.character)]), names(data_set[sapply(data_set, is.fac
tor)]))
     for (f_name in setdiff(text_features, features_to_ignore)) {
          f_vector <- as.character(data_set[,f_name])
          # treat as raw text if data over minimum_precent_unique unique
          if (length(unique(as.character(f_vector))) > (nrow(data_set) * minimum_unique_threshold)) {
               data_set[,paste0(f_name, '_word_count')] <- sapply(strsplit(f_vector, " "), length)
               data_set[,paste0(f_name, '_character_count')] <- nchar(as.character(f_vector))
               data_set[,paste0(f_name, '_first_word')] <- sapply(strsplit(as.character(f_vector), " "), `[`,
1)
               # remove orginal field
               data_set[,f_name] <- NULL
          }
     }
     return(data_set)
}
```

Now, let's load the Auto MPG Data Set (https://archive.ics.uci.edu/ml/datasets/Auto+MPG). This is a simple data set but requires the above function calls as it contains missing data, character-based variables, and numerical data. Let's load it in memory and run our pipeline functions:

```
AutoMpg_data <-  read.csv("http://mlr.cs.umass.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data", na.st
rings = '?', header=FALSE, sep="", as.is=TRUE, col.names = c("mpg", "cylinders", "displacement", "horsepower",
"weight", "acceleration", "model", "origin", "car_name"), stringsAsFactors = FALSE)

AutoMpg_data <- Get_Free_Text_Measures(data_set = AutoMpg_data, minimum_unique_threshold=0.5)
AutoMpg_data <- Impute_Features(data_set = AutoMpg_data, use_mean_instead_of_0 = FALSE)
```

```
## [1] "mpg"
## [1] "cylinders"
## [1] "displacement"
## [1] "horsepower"
## [1] "weight"
## [1] "acceleration"
## [1] "model"
## [1] "origin"
## [1] "car_name_word_count"
## [1] "car_name_character_count"
## [1] "car_name_first_word"
```

Let's take a quick look at the data:

```
str(AutoMpg_data)
```

```
## 'data.frame':    398 obs. of  11 variables:
##  $ mpg                     : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders               : num  8 8 8 8 8 8 8 8 8 ...
##  $ displacement            : num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower              : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight                  : num  3504 3693 3436 3433 3449 ...
##  $ acceleration            : num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ model                   : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin                  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ car_name_word_count     : num  3 3 2 3 2 3 2 3 2 3 ...
##  $ car_name_character_count: num  25 17 18 13 11 16 16 17 16 18 ...
##  $ car_name_first_word     : chr  "chevrolet" "buick" "plymouth" "amc" ...
```

Let's use the k-means model to discover the relationship between `acceleration` and `weight` of vehicles using 3 clusters.

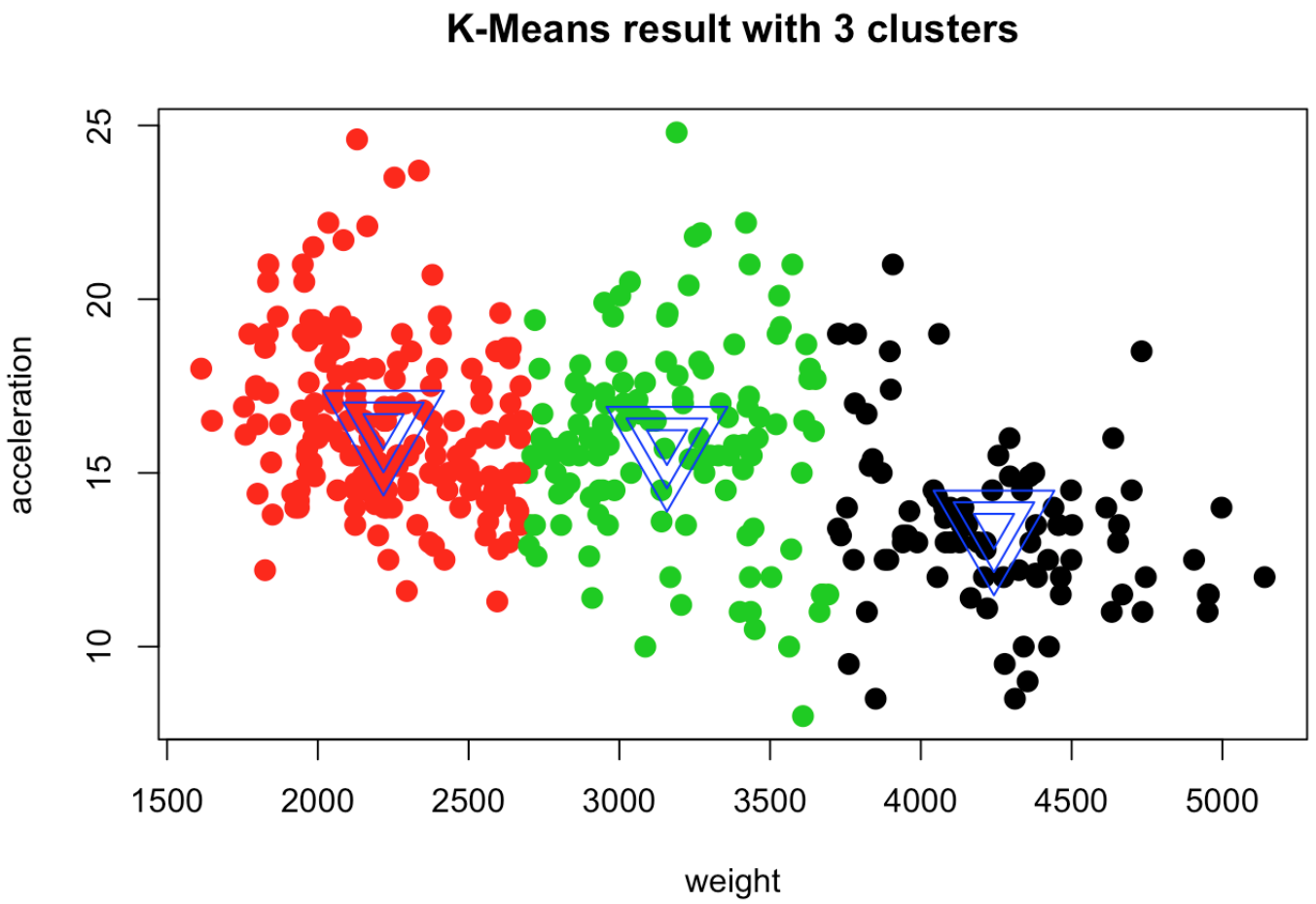
```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
set.seed(1234)
km1 = kmeans(x = select(AutoMpg_data, weight, acceleration), centers = 3)

# Plot results
plot(select(AutoMpg_data, weight, acceleration),
     col =km1$cluster, main="K-Means result with 3 clusters",
     pch=20, cex=2)

# find each cluster's centroids
points(km1$centers, pch=6, col='blue', cex=6)
points(km1$centers, pch=6, col='blue', cex=4)
points(km1$centers, pch=6, col='blue', cex=2)
```



`K-means` did split the data into three groups. Unfortunately, the use of k-means here is dimished due to the obvious linear relationship between both variables.

To make this clustering discovery more interesting, we're going to transform our data set using the `car_name_first_word` variable which is none other than the brand of each car:

```
unique(AutoMpg_data$car_name_first_word)
```

```
##  [1] "chevrolet"     "buick"         "plymouth"      "amc"
##  [5] "ford"          "pontiac"       "dodge"         "toyota"
##  [9] "datsun"        "volkswagen"    "peugeot"       "audi"
## [13] "saab"          "bmw"           "chevy"         "hi"
## [17] "mercury"       "opel"          "fiat"          "oldsmobile"
## [21] "chrysler"      "mazda"         "volvo"         "renault"
## [25] "toyouta"       "maxda"         "honda"         "subaru"
## [29] "chevroelt"     "capri"         "vw"            "mercedes-benz"
## [33] "cadillac"      "mercedes"      "vokswagen"     "triumph"
## [37] "nissan"
```

Let's group the data by brand into a new data set called `brand_set` and average all the data by brand. We also use the brand name as a row name (this will become apparent later on):

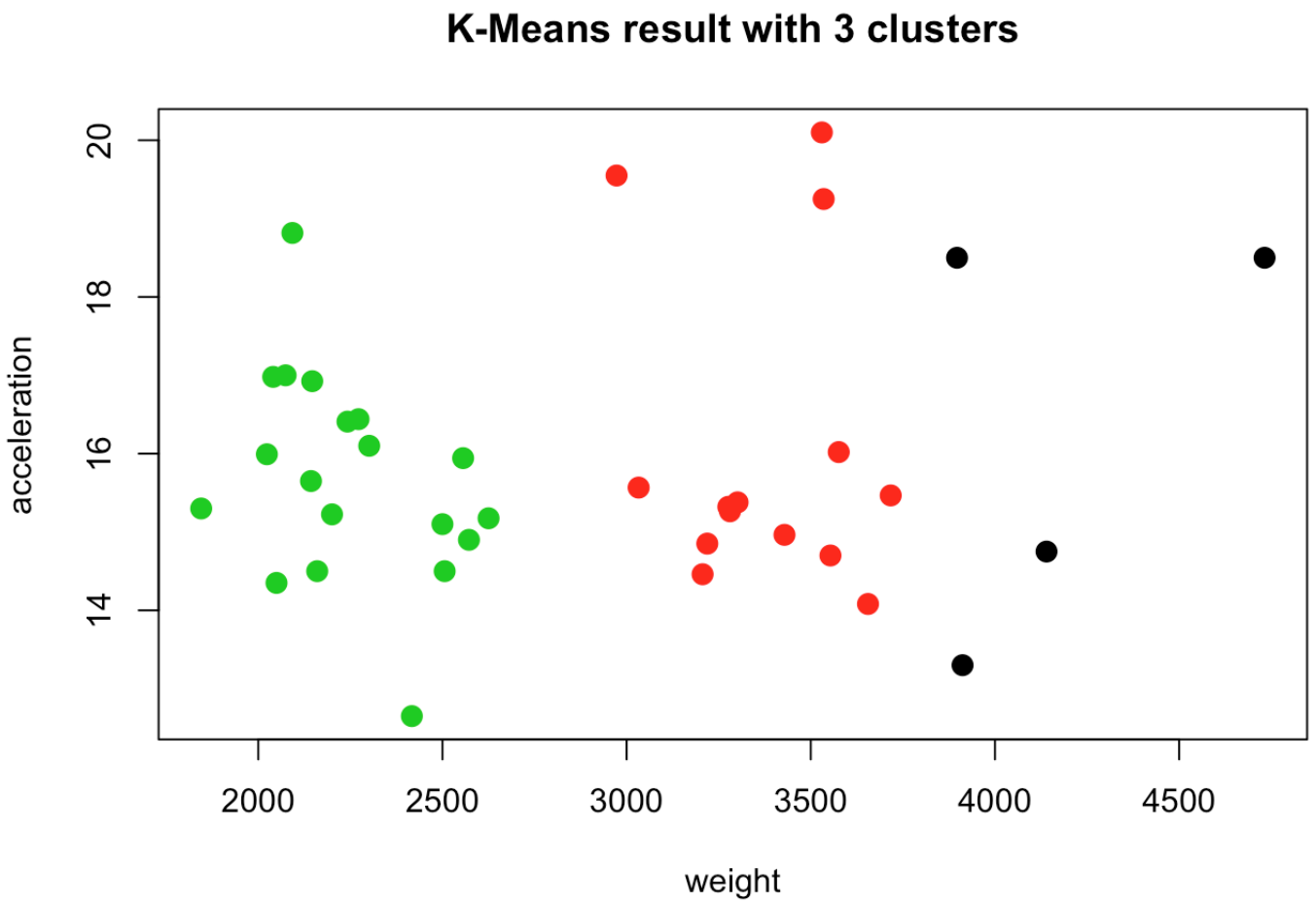
```
brand_set <- select(AutoMpg_data, weight, acceleration, car_name_first_word) %>%
     group_by(car_name_first_word) %>% summarize_each(funs(mean)) %>% data.frame

row.names(brand_set) <- brand_set$car_name_first_word

brand_set <- dplyr::select(brand_set, -car_name_first_word)

set.seed(1234)
km1 = kmeans(x = brand_set, centers = 3)

# Plot results
plot(brand_set,
     col =km1$cluster, main="K-Means result with 3 clusters",
     pch=20, cex=2)
```



Looking at the grouped data by brand we clearly see three clusters. The first cluter (greens) is clearly away for the other data. Overall, this is hard to read beyond seeing some form of clustering.

Let's use the factoextra (https://cran.r-project.org/web/packages/factoextra/index.html) library. This is why we replaced the row names with the actual brand!

```
# install.packages('factoextra')
library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 3.2.5
```

```
## Loading required package: ggplot2
```

```
set.seed(1234)
km1 = kmeans(x = brand_set, centers = 3)
print(km1)
```

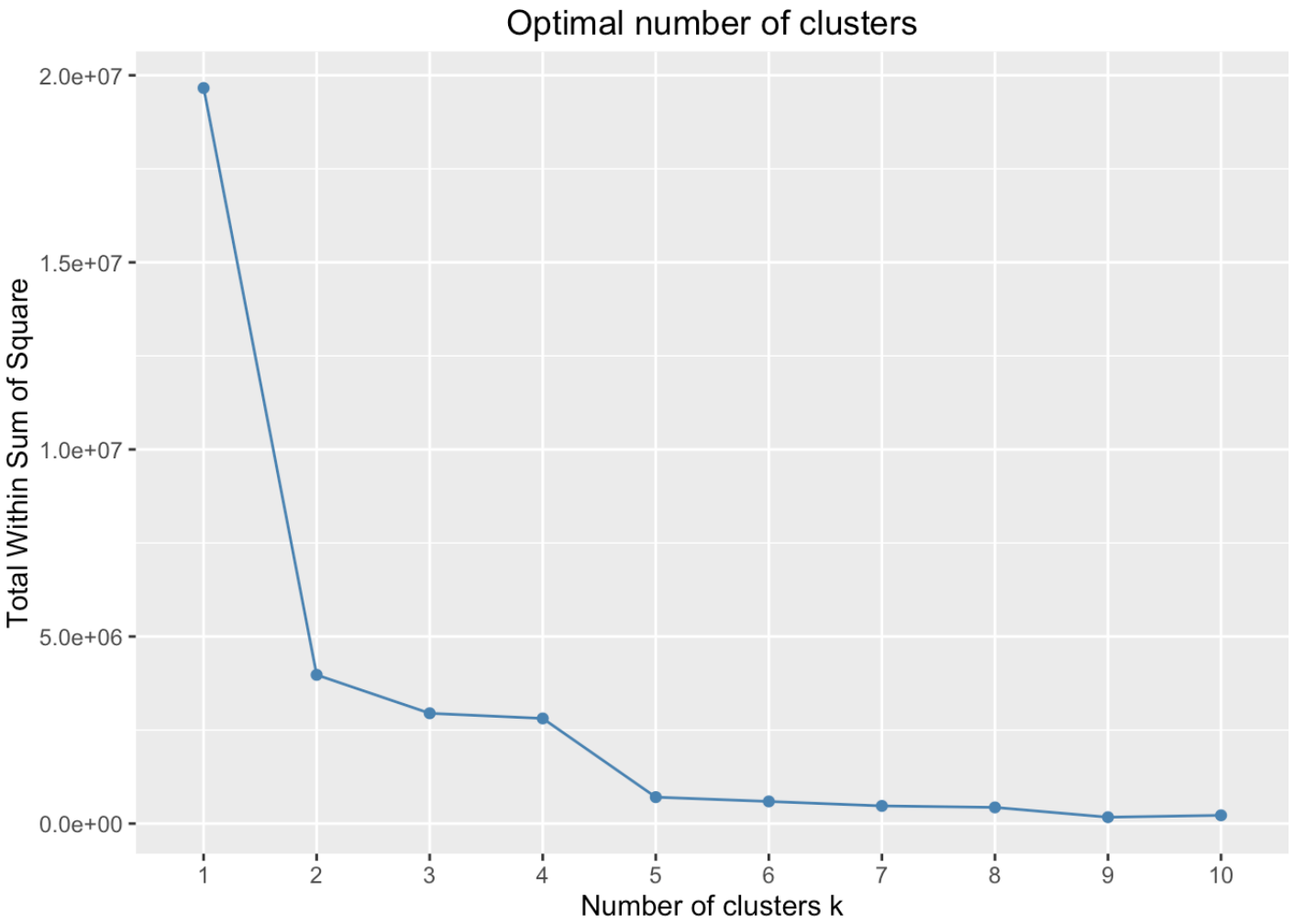
```
## K-means clustering with 3 clusters of sizes 4, 14, 19
##
## Cluster means:
##     weight acceleration
## 1 4170.250     16.26250
## 2 3377.402     16.06959
## 3 2250.898     15.68187
##
## Clustering vector:
##           amc          audi           bmw         buick      cadillac
##             2             3             3             2             1
##         capri     chevroelt     chevrolet         chevy      chrysler
##             3             1             2             2             1
##        datsun         dodge          fiat          ford            hi
##             3             2             3             2             1
##         honda         maxda         mazda      mercedes mercedes-benz
##             3             3             3             2             2
##       mercury        nissan    oldsmobile          opel       peugeot
##             2             3             2             3             2
##      plymouth       pontiac       renault          saab        subaru
##             2             2             3             3             3
##        toyota       toyouta       triumph     vokswagen    volkswagen
##             3             3             3             3             3
##         volvo            vw
##             2             3
##
## Within cluster sum of squares by cluster:
## [1] 457857.8 676136.5 885185.0
##  (between_SS / total_SS =  89.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
fviz_cluster(km1, data = brand_set)
```

Wow, right? Clear patterns in the data. We can see that European and Asian vehicles in this data set are lighter than American ones.

Another great tool in `factoextra` is the ability to advise on how many clusters to use.

```
set.seed(1234)
fviz_nbclust(brand_set, kmeans, method = "wss")
```



Seems that `fviz_nbclust` is recommending `4` clusters using the `within cluster sums of squares` method. Go ahead, try different sizes and methods.

I learnd about `factoextra` through the following article - Partitioning cluster analysis: Quick start guide - Unsupervised Machine Learning (http://www.sthda.com/english/wiki/partitioning-cluster-analysis-quick-start-guide-unsupervised-machine-learning) - great tips and more lessons there!