R Lab: K-Means Clustering

In this lab, we use the 'NbClust' and 'kmeans' R functions to implement k-means clustering on the lab data. We first load the R libraries we will be using. The only new R library needed is NbClust (for the 'NbClust' function), since the 'kmeans' function is available in the base R installation.

```
library(dplyr)
library(caret)
library(NbClust)
```

K-Means Clustering

Before we implement k-means clustering, we follow the same data pre-processing steps as in the previous labs.

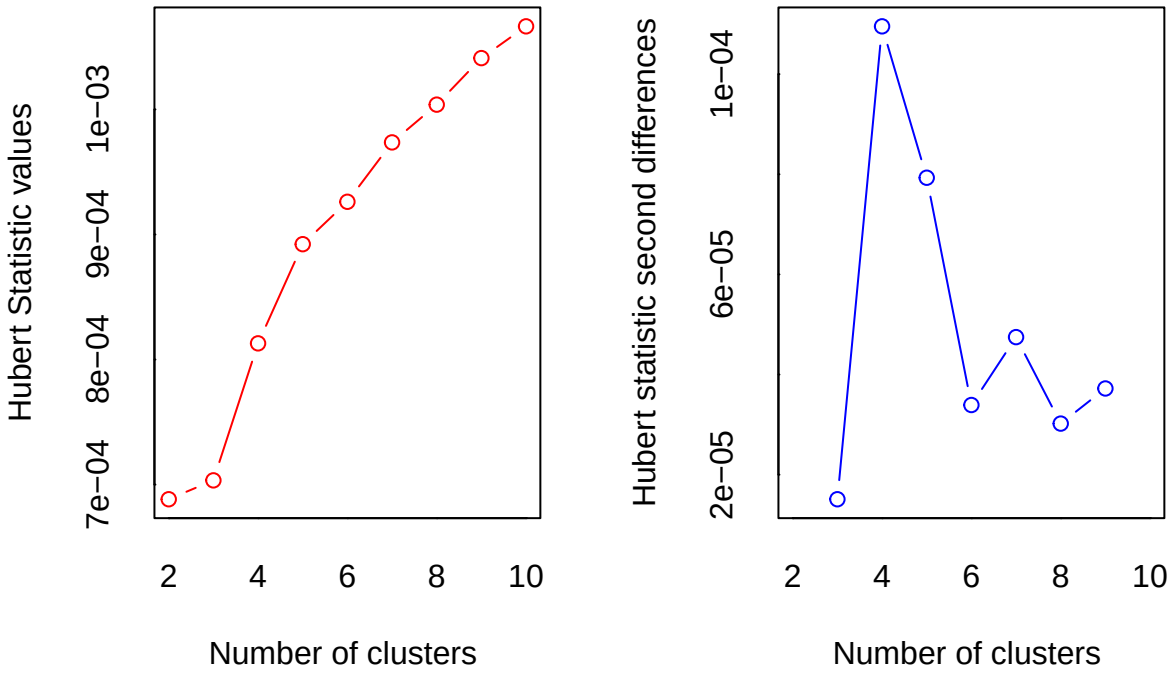
```
data <- read.csv("lab_8_data.csv")
training_ind <- createDataPartition(data$lodgepole_pine,
                                    p = 0.75,
                                    list = FALSE,
                                    times = 1)
training_set <- data[training_ind, ]
test_set <- data[-training_ind, ]

top_20_soil_types <- training_set %>%
  group_by(soil_type) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  select(soil_type) %>%
  top_n(20)

training_set$soil_type <- ifelse(training_set$soil_type %in% top_20_soil_types$soil_type,
                                 training_set$soil_type,
                                 "other")

training_set$wilderness_area <- factor(training_set$wilderness_area)
training_set$soil_type <- factor(training_set$soil_type)

onehot_encoder <- dummyVars(~ wilderness_area + soil_type,
                            training_set[, c("wilderness_area", "soil_type")],
                            levelsOnly = TRUE,
                            fullRank = TRUE)

onehot_enc_training <- predict(onehot_encoder,
                               training_set[, c("wilderness_area", "soil_type")])

training_set <- cbind(training_set, onehot_enc_training)

test_set$soil_type <- ifelse(test_set$soil_type %in% top_20_soil_types$soil_type,
                             test_set$soil_type,
                             "other")
```

```
test_set$wilderness_area <- factor(test_set$wilderness_area)
test_set$soil_type <- factor(test_set$soil_type)

onehot_enc_test <- predict(onehot_encoder, test_set[, c("wilderness_area", "soil_type")])
test_set <- cbind(test_set, onehot_enc_test)

test_set[, -c(11:13)] <- scale(test_set[, -c(11:13)],
                        center = apply(training_set[, -c(11:13)], 2, mean),
                        scale = apply(training_set[, -c(11:13)], 2, sd))
training_set[, -c(11:13)] <- scale(training_set[, -c(11:13)])

training_features <- array(data = unlist(training_set[, -c(11:13)]),
                  dim = c(nrow(training_set), 33))
training_labels <- array(data = unlist(training_set[, 13]),
                  dim = c(nrow(training_set)))

test_features <- array(data = unlist(test_set[, -c(11:13)]),
                  dim = c(nrow(test_set), 33))
test_labels <- array(data = unlist(test_set[, 13]),
                  dim = c(nrow(test_set)))
```

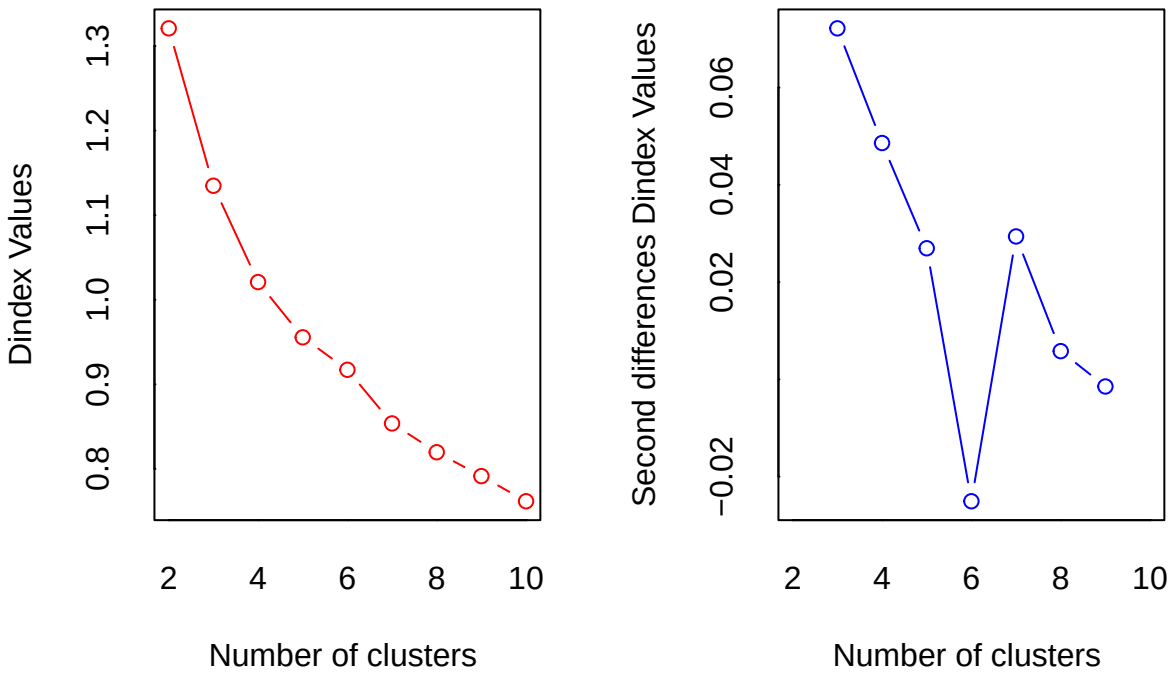
Note, as discussed in the live session, the scaling of the training and test sets above is critical for k-means clustering. Now that the features have been scaled, we implement k-means in R using the 'NbClust' and 'k-means' functions. Let's see if the observations can be clustered based on their horizontal distance to important landmarks (horizontal_distance_to_hydrology, horizontal_distance_to_roadways, and horizontal_distance_to_fire_points). In practice, we can cluster based on any features we think are relevant; this is where the subjective part of unsupervised learning comes into play, since our analysis is often guided by subject matter knowledge and expertise.

We first use the 'NbClust' function to determine if there is an optimal number of clusters according to the 26 criteria used by this function. We use a random sample of 1000 rows of the original training features, since the 'NbClust' function can take a long time to run on large datasets. We set the seed for random number generation, so the exact same results will be obtained when run on your machine (otherwise, the results may be different, since the sample of 1000 rows may be different).

```
set.seed(123)
nc <- NbClust(training_features[sample(nrow(training_features), 1000), c(4, 6, 10)],
              min.nc = 2, max.nc = 10, method = "kmeans")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 5 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 10 proposed 4 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
##
##                   ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  4
##
##
## *******************************************************************
```

As can be seen in the output of the 'NbClust' function, the majority of the 26 criteria voted for 4 clusters to use for k-means. Now, to obtain the 4 clusters, we use the 'kmeans' function with the 'centers' argument set to 4.

```
km_clusters <- kmeans(training_features[, c(4, 6, 10)], centers = 4)
```

After calculating the clusters, we can append the cluster labels to the training features. Now, we can use the cluster labels as a categorical feature in a supervised learning model (such as a dense feed-forward neural network). As with any other categorical feature, we need to encode the cluster label (such as using one-hot encoding) before applying a supervised learning model.

```
cluster_number <- data.frame(cluster_number = km_clusters$cluster)
training_features <- cbind(training_features, cluster_number)
head(training_features)
```

```
##            1          2          3           4          5          6         7
## 1 -1.6167501 -0.5639102 -0.9654917 -0.415395248 -0.4910363 -1.0494217 0.6967985
## 2 -0.2846512 -0.6177651 -0.2860701 -1.139860496 -0.7502340  1.2888698 0.9225641
## 3 -0.8153248 -0.1959014  0.2574672 -0.196149186  0.5630343 -0.4928993 1.1859574
## 4 -0.3640718 -0.2677080 -0.8296074 -0.005500437 -0.6811146  0.5957815 0.7720537
## 5 -0.4976427 -0.7434267  0.8010045 -0.853887372 -0.4564766  0.3399607 0.9601917
## 6 -0.7756145 -0.2946355  0.6651202 -1.139860496 -0.7847936  0.2457110 1.3364678
##             8          9         10         11         12         13
## 1  0.29679857 -0.2909338  2.6638157 -0.2249868 -0.8854354 -0.2562993
## 2 -0.11362446 -0.7638438  2.9761371 -0.2249868 -0.8854354 -0.2562993
## 3  0.39940433 -0.8163894 -0.8353962 -0.2249868 -0.8854354 -0.2562993
## 4  0.60461585 -0.2121154  0.7989844 -0.2249868 -0.8854354 -0.2562993
## 5 -1.29359068 -1.5783000  0.2023900 -0.2249868 -0.8854354 -0.2562993
## 6 -0.06232158 -1.2630266  0.1910191 -0.2249868 -0.8854354 -0.2562993
##            14          15         16          17         18         19
## 1 -0.04632411 -0.03500164 -0.5012998 -0.09040308 -0.2363875 -0.2204473
## 2 -0.04632411 -0.03500164 -0.5012998 -0.09040308  4.2296958 -0.2204473
## 3 -0.04632411 -0.03500164 -0.5012998 -0.09040308  4.2296958 -0.2204473
## 4 -0.04632411 -0.03500164 -0.5012998 -0.09040308 -0.2363875 -0.2204473
## 5 -0.04632411 -0.03500164  1.9945090 -0.09040308 -0.2363875 -0.2204473
## 6 -0.04632411 -0.03500164  1.9945090 -0.09040308 -0.2363875 -0.2204473
##           20         21          22          23          24          25
## 1 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
## 2 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
## 3 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
## 4 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
## 5 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
## 6 -0.3096418 -0.2930312 -0.04795369 -0.06439589 -0.01236832 -0.01749279
##           26         27        28         29          30         31          32
## 1 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
## 2 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
## 3 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
## 4 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
## 5 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
## 6 -0.1604046 -0.1495369 -0.143547 -0.1188071 -0.04795369 -0.1025186 -0.01749279
##            33 cluster_number
## 1 -0.04288122              1
## 2 -0.04288122              1
## 3 -0.04288122              3
## 4 -0.04288122              4
## 5 -0.04288122              4
## 6 -0.04288122              3
```

In this lab, we used the 'NbClust' and 'kmeans' functions to implement k-means clustering. However, several

different methods for implementing clustering exist and are also implemented in R. Look into the different methods, as well as other arguments, that are implemented in the 'NbClust' function discussed above.

Exercises

1) Why is it good practice to center and scale before applying k-means clustering?
2) Print the cluster sizes and centers to the R console. Include a screenshot of the output.
3) Use the aggregate() function to calculate the mean of each variable within each cluster. Include a screenshot of the output.
4) Assign the observations in test_features to the clusters found using the sample of the observations in training_features above. Include a screenshot of the output. Hint: a function in the 'clue' R library can be used for this.