week1

Rajesh Kalakoti

2023-08-03

- Packages
  - devtools
  - tidyverse
  - here

```r
# Include the script from the R directory
project_path <- here()
source(here("R", "utils.R"))
source(here("R","distance_functions.R"))
```

Clustering

Given a clustering $C = \{C_1, C_2, \dots, C_k\}$, we need some scoring function that evaluates its quality or goodness. This sum of squared errors scoring function is defined as:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{i:C(i)=k} \|x_i - \bar{x}_k\|^2$$

The goal is to find the clustering that minimizes:

$$C^* = \arg\min_{C} \{W(c)\}$$

K-means employs a greedy iterative approach to find a clustering that minimizes loss function.

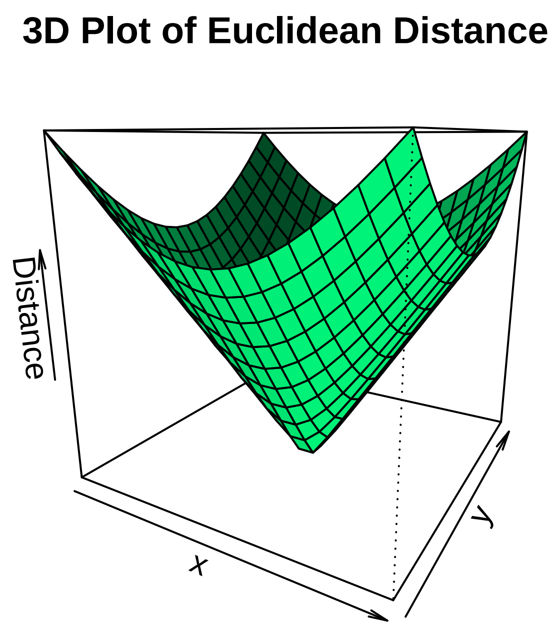
Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```r
euclidean_dist <- function(point1, point2) {
  squared_diff <- (point1 - point2)^2
  sqrt(sum(squared_diff))
}

x <- y <- seq(-1, 1, length = 20)
grid <- expand.grid(x = x, y = y)  # Create a grid of points
z <- matrix(0, nrow = length(x), ncol = length(y))  # Initialize the z matrix

for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    z[i, j] <- euclidean_dist(c(x[i], y[j]), c(0, 0))
  }
}
persp(x, y, z,
```

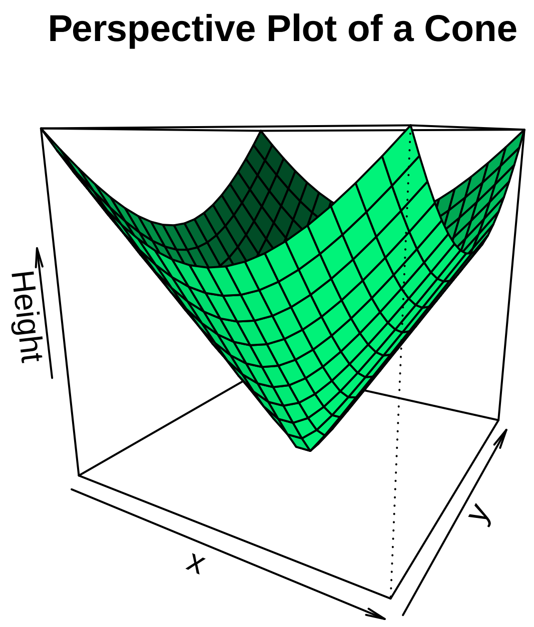
Algorithm 1: K-means Algorithm

```
Data: $D, k, \varepsilon$
1  K-means($D, k, \varepsilon$):
2  $t \leftarrow 0$;
3  Randomly initialize $k$ centroids: $\mu_1^t, \mu_2^t, \ldots, \mu_n^t \in \mathbb{R}^d$;
4  repeat
5      $t \leftarrow t + 1$;
6      $C_i \leftarrow \emptyset$ for all $i = 1, \ldots, k$
7      /* Cluster assignment step */
8      for $x_j \in D$ do
9          $i^* \leftarrow \text{argmin}_i\{||x_j - \mu_i^{t-1}||^2\}$;
10         /* assign $x_j$ to closest centroid */
11         $C_{i^*} \leftarrow C_{i^*} \cup \{x_j\}$;
12     end
13     for $i = 1, .., k$ do
14         $\mu_i^t \leftarrow \frac{1}{|C_i|} \sum_{x_j \in C_i} X_j$
15     end
16 until $\sum_{i=1}^{k} ||\mu_i^t - \mu_i^{t-1}||^2 \leq \varepsilon$;
```

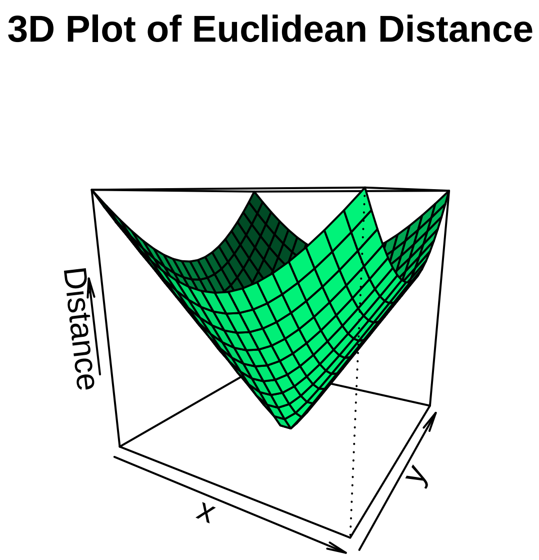
```r
      main = "3D Plot of Euclidean Distance",
      zlab = "Distance",
      theta = 30, phi = 15,
      col = "springgreen", shade = 0.5)
```

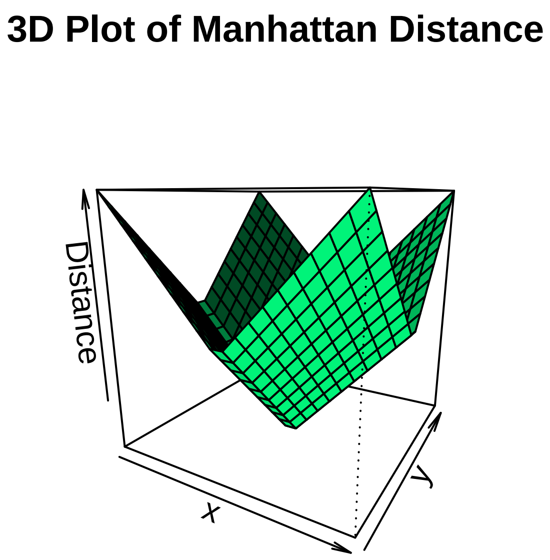


```r
cone1 <- function(x, y){
sqrt(x^2+y^2)
}

x <- y <- seq(-1, 1, length= 20)
z <- outer(x, y, cone1)

persp(x, y, z,
main="Perspective Plot of a Cone",
zlab = "Height",
theta = 30, phi = 15,
col = "springgreen", shade = 0.5)
```



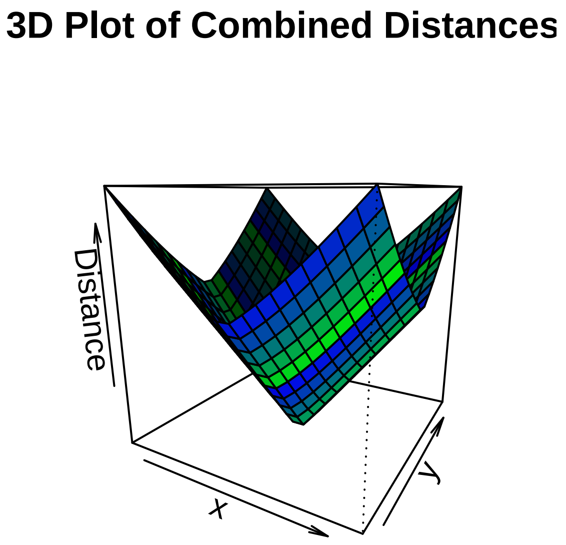
```r
euclidean_dist <- function(point1, point2) {
  squared_diff <- (point1 - point2)^2
  sqrt(sum(squared_diff))
}

manhattan_distance <- function(point1, point2) {
  if (length(point1) != length(point2)) {
    stop("Both points should have the same number of dimensions.")
  }

  abs_diff <- abs(point1 - point2)
  distance <- sum(abs_diff)
  return(distance)
}

x <- y <- seq(-1, 1, length = 20)
grid <- expand.grid(x = x, y = y)  # Create a grid of points
z_euclidean <- matrix(0, nrow = length(x), ncol = length(y))  # Initialize the z matrix for Euclidean d
z_manhattan <- matrix(0, nrow = length(x), ncol = length(y))   # Initialize the z matrix for Manhattan

for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    z_euclidean[i, j] <- euclidean_dist(c(x[i], y[j]), c(0, 0))
    z_manhattan[i, j] <- manhattan_distance(c(x[i], y[j]), c(0, 0))
  }
}

# Create a layout of subplots to show both Euclidean and Manhattan distances
par(mfrow = c(1, 2))

# Plot for Euclidean distance
persp(x, y, z_euclidean,
      main = "3D Plot of Euclidean Distance",
      zlab = "Distance",
      theta = 30, phi = 15,
      col = "springgreen", shade = 0.5)

# Plot for Manhattan distance
persp(x, y, z_manhattan,
      main = "3D Plot of Manhattan Distance",
      zlab = "Distance",
      theta = 30, phi = 15,
      col = "springgreen", shade = 0.5)
```





```r
# Reset the layout
par(mfrow = c(1, 1))
```

```r
euclidean_dist <- function(point1, point2) {
  squared_diff <- (point1 - point2)^2
  sqrt(sum(squared_diff))
}

manhattan_distance <- function(point1, point2) {
  if (length(point1) != length(point2)) {
    stop("Both points should have the same number of dimensions.")
  }

  abs_diff <- abs(point1 - point2)
  distance <- sum(abs_diff)
  return(distance)
}

x <- y <- seq(-5, 5, length = 20)
grid <- expand.grid(x = x, y = y)  # Create a grid of points

z_euclidean <- matrix(0, nrow = length(x), ncol = length(y))  # Initialize the z matrix for Euclidean d
z_manhattan <- matrix(0, nrow = length(x), ncol = length(y))   # Initialize the z matrix for Manhattan

for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    z_euclidean[i, j] <- euclidean_dist(c(x[i], y[j]), c(0, 0))
    z_manhattan[i, j] <- manhattan_distance(c(x[i], y[j]), c(0, 0))
  }
}

# Combine the distances and choose different colors for each
combined_distances <- z_euclidean + z_manhattan
color_palette <- colorRampPalette(c("blue", "green"))(100)  # Choose colors for mapping distances

# Create a layout of subplots
layout(matrix(c(1, 2), nrow = 1))

# Plot both distances on the same 3D plane with different colors
persp(x, y, combined_distances,
      main = "3D Plot of Combined Distances",
      zlab = "Distance",
      theta = 30, phi = 15,
      col = color_palette, shade = 0.5)

# Reset the layout
layout(1)
```



```r
# Include the script from the R directory
project_path <- here()
source(here("R", "utils.R"))
```

```r
?entropy
```

```
## No documentation for 'entropy' in specified packages and libraries:
## you could try '??entropy'
```