Variant of k-means for acceleration and better convergence

Piseth KHENG, Borachhun YOU

24 October 2022

Exercise 1: `k-means++` algorithm

1. Programming `k-means++` algorithm

`k-means++` is an algorithm proposed by David Arthur and Sergei Vassilvitskii with the goal of improving the convergence and speed of the `k-means` algorithm, and it does so by carefully choosing the center of the clusters at the initial step. With `k-means`, it initially chooses the centers uniformly at random from the data points. In contrast, `k-means++` chooses the centers as followed:

i. Choose one center c_1 uniformly at random from the set of data points $\mathcal{X}$
ii. Choose the next center c_i by selecting $x' \in \mathcal{X}$ with weighted probability $\frac{D(x')^2}{\sum_{x \in \mathcal{X}} D(x)^2}$, where $D(x)$ is the shortest distance from x to the closest center that has already been chosen
iii. Repeat step ii until a total of k centers are chosen, where k is a given number of clusters.

After choosing the centers, `k-means++` proceeds the same computations as `k-means`.

```
kmeanspp <- function(X, k) {
  X <- as.matrix(X)
  X_row <- nrow(X)

  # choose first center uniformly at random
  center_index <- sample(1:X_row, size=1)

  for (i in 2:k) {

    # calculate squared distance to closest chosen center
    D2 <- c()
    for (d in 1:X_row) {
      data_centers_dist_sq <- c()
      for (c in center_index) {
        data_centers_dist_sq <- c(data_centers_dist_sq, sum((X[d,]-X[c,])^2))
      }
      D2 <- c(D2, min(data_centers_dist_sq))
    }

    # choose a new center
    center_index <- c(center_index, sample(1:X_row, size=1, prob=(D2/sum(D2))))
  }

  # run kmeans with initialized centers
  return(kmeans(X, centers=X[center_index,]))
}
```

2. Simulate `NORM-10` and `NORM-25` datasets

We now simulate 2 datasets, `NORM-10` and `NORM-25`, for evaluating the performance of the `k-means++` algorithm. To generate the datasets, we choose 10 (or 25) "real" centers uniformly at random from a hypercube of side length 500. We then add points from Gaussian distributions of variance 1 around each real center.

- For `NORM-10`, we generate 1000 data around each of the 10 centers of dimension 5
- For `NORM-25`, we generate 400 data around each of the 25 centers of dimension 15.

```
# n = number of centers
# dim = dimension of data
# pts = number of points around each center
NORM <- function(n, dim, pts) {
  set.seed(NULL)

  # choose true centers
  center_coor <- c()
  for (i in 1:(n*dim)) {
    center_coor <- c(center_coor, runif(1, min=0, max=500))
  }
  true_centers <- matrix(center_coor, nrow=n, ncol=dim, byrow=TRUE)

  # add points around centers
  res <- c()
  for (ct_row in 1:n) {
    for (i in 1:pts) {
      for (ct_col in 1:dim) {
        res <- c(res, rnorm(1, mean=true_centers[ct_row, ct_col], sd=1))
      }
    }
  }

  return(matrix(res, ncol=dim, byrow=TRUE))
}

`NORM-10` <- NORM(n=10, dim=5, pts=1000)
`NORM-25` <- NORM(n=25, dim=15, pts=400)
```

3. `k-means` and `k-means++` comparison

With `k-means` (and thus `k-means++` as well), the problem involves minimizing

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2$$

where $\mathcal{X}$ is the set of data points and $\mathcal{C}$ is the set of cluster centers.

We now use the value of ϕ and the execution time of both algorithms to compare the performance between the two.

```
phi <- function(X, centers) {
  res <- 0
  for (x_row in 1:nrow(X)) {
    to_min <- c()
    for (c_row in 1:nrow(centers)) {
      to_min <- c(to_min, sum((X[x_row,] - centers[c_row,])^2))
    }
    res <- res + min(to_min)
  }
  return(res)
}
```

```
perf <- function(dataset) {
  T_km <- c()
  T_kmpp <- c()

  phi_km <- c()
  phi_kmpp <- c()

  for (k in c(10,25,50)) {
    for (i in 1:20) {
      Sys.time() -> begin_km
      km <- kmeans(dataset, k)
      Sys.time() -> end_km

      Sys.time() -> begin_kmpp
      kmpp <- kmeanspp(dataset, k)
      Sys.time() -> end_kmpp

      T_km <- c(T_km, end_km-begin_km)
      T_kmpp <- c(T_kmpp, end_kmpp-begin_kmpp)
      phi_km <- c(phi_km, phi(dataset, km$centers))
      phi_kmpp <- c(phi_kmpp, phi(dataset, kmpp$centers))
    }
  }

  return(list(
    T_km_10 = T_km[1:20],
    T_km_25 = T_km[21:40],
    T_km_50 = T_km[41:60],

    T_kmpp_10 = T_kmpp[1:20],
    T_kmpp_25 = T_kmpp[21:40],
    T_kmpp_50 = T_kmpp[41:60],

    phi_km_10 = phi_km[1:20],
    phi_km_25 = phi_km[21:40],
    phi_km_50 = phi_km[41:60],

    phi_kmpp_10 = phi_kmpp[1:20],
    phi_kmpp_25 = phi_kmpp[21:40],
    phi_kmpp_50 = phi_kmpp[41:60]
  ))
}
```

```
perf_NORM_10 <- perf(`NORM-10`)
perf_NORM_25 <- perf(`NORM-25`)
```

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	9.0970119×10^7	5.0073554×10^4	1.653474×10^7	5.0073554×10^4	0.0059944	2.8135006
25	3.0814732×10^6	4.1324281×10^4	4.0913514×10^4	4.0840309×10^4	0.0156347	12.0267773
50	3.3167689×10^4	3.2666239×10^4	3.2193716×10^4	3.207215×10^4	0.0282081	41.2566333

Table 1: Experimental results on the *Norm-10* dataset (n = 10000, d = 5)

Based on the performance results of table 1 and table 2, in terms of the average value of ϕ, we can see that `k-means++` performs drastically better than `k-means` (k = 10 and 25 for `NORM-10`, k = 25 and 50 for `NORM-25`). As for the minimum ϕ, it also shows that `k-means++` has a significantly better

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	1.2083996×10^9	1.0963251×10^9	1.0231659×10^9	1.001124×10^9	0.0101364	2.9782167
25	4.2612183×10^8	1.4938423×10^5	1.9609426×10^8	1.4938423×10^5	0.0181762	12.9584825
50	6.564006×10^7	1.4155882×10^5	1.4132443×10^5	1.4126846×10^5	0.034395	44.911974

Table 2: Experimental results on the *Norm-25* dataset (n = 10000, d = 15)

performance than `k-means`, with k = 10 for `NORM-10` and k = 25 for `NORM-25`, while the two algorithm performs approximately the same for the other values of k. Finally, regarding the execution time, `k-means` performs a lot faster than `k-means++` in all cases. This may be a result of having the centers initialization step of `k-means++` unoptimized compared to the optimized `k-means` function in r library.

Exercise 2: `iris` dataset

1. Apply `k-means++`, `k-means` and `Mclust` on `iris` dataset

The `iris` dataset gives measurements of 4 variables (sepal length and width, petal length and width) of 50 flowers from each of 3 species of iris (Setosa, Versicolor, Virginica).

```
data_iris <- iris[,1:4]
iris_class <- iris[,5]
```

```
iris.kmpp <- kmeanspp(data_iris, 3)
kmpp_class <- iris.kmpp$cluster
```

```
iris.km <- kmeans(data_iris, 3)
km_class <- iris.km$cluster
```

```
library(mclust)
```

```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

```
iris.mclust <- Mclust(data_iris, G=3)
summary(iris.mclust)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 3 components:
##
##  log-likelihood   n df       BIC       ICL
##        -186.074 150 38 -562.5522 -566.4673
##
## Clustering table:
##  1  2  3
## 50 45 55
```

```
m_class <- iris.mclust$classification
```

```r
table(`k-means++`=kmpp_class, iris=iris_class)
```

```
##          iris
## k-means++ setosa versicolor virginica
##         1      0         48        14
##         2     50          0         0
##         3      0          2        36
```

```r
table(`k-means`=km_class, iris=iris_class)
```

```
##        iris
## k-means setosa versicolor virginica
##       1     50          0         0
##       2      0          2        36
##       3      0         48        14
```

```r
table(Mclust=m_class, iris=iris_class)
```

```
##       iris
## Mclust setosa versicolor virginica
##      1     50          0         0
##      2      0         45         0
##      3      0          5        50
```

According to the cluster comparison tables above, we can see that both `k-means` and `k-means++` have the same result, by correctly clustering Setosa species and making 2 and 14 mistakes with Versicolor and Virginica species respectively. As for `Mclust`, it managed to perform better, by only making 5 mistakes with Versicolor species.

2. Visualize the different partitions on PCA plan

```r
library(FactoMineR)

iris_scale <- scale(data_iris, center=TRUE, scale=FALSE)
iris.pca <- PCA(iris_scale, scale.unit=FALSE, ncp=4, graph=FALSE)
```

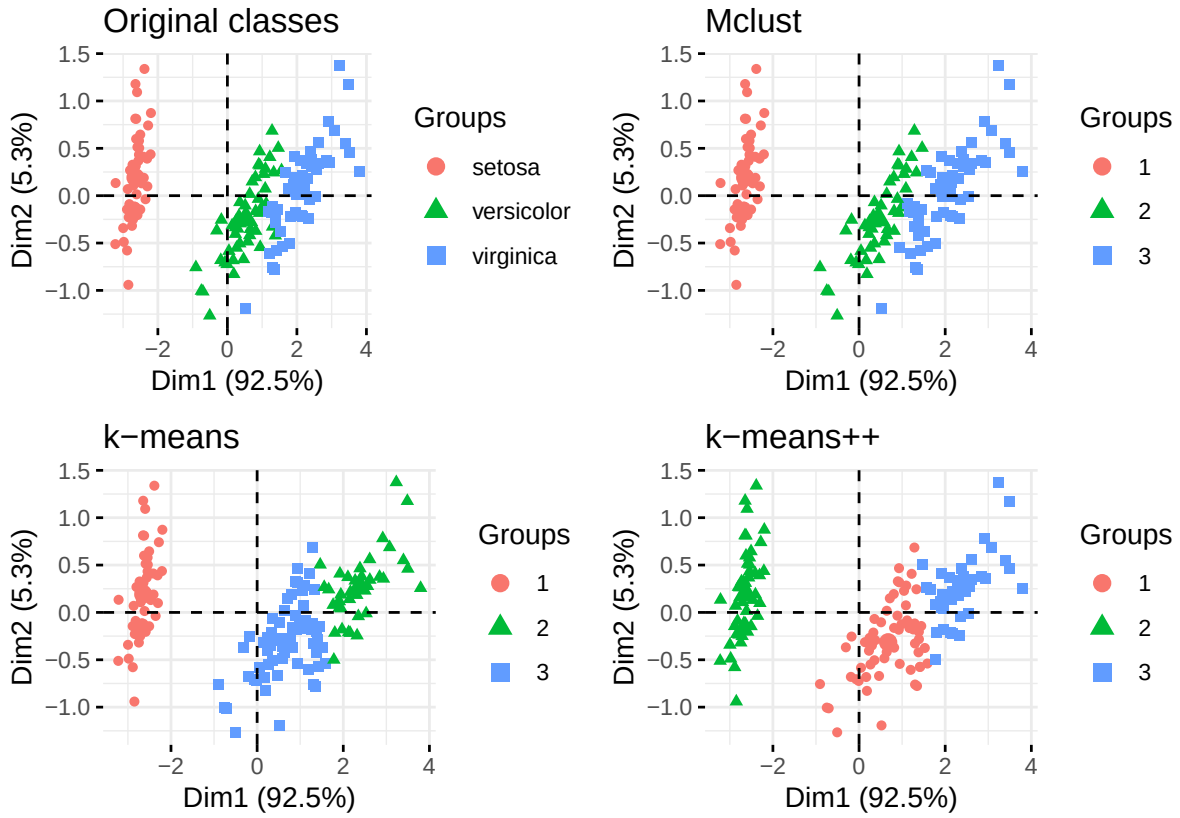
```r
library(factoextra)
```

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

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
iris.plot <- fviz_pca_ind(iris.pca, label="none",
                          habillage = as.factor(iris_class), title="Original classes")
iris.plot.kmeanspp <- fviz_pca_ind(iris.pca, label="none",
                                   habillage = as.factor(kmpp_class), title = "k-means++")
iris.plot.kmeans <- fviz_pca_ind(iris.pca, label="none",
                                 habillage = as.factor(km_class), title = "k-means")
iris.plot.mclust <- fviz_pca_ind(iris.pca, label="none",
                                 habillage = as.factor(m_class), title = "Mclust")
```

```
library(gridExtra)
grid.arrange(
  iris.plot,
  iris.plot.mclust,
  iris.plot.kmeans,
  iris.plot.kmeanspp,
  ncol=2
)
```



3. Comment

Based on the plotted graphs above, we can see that the partition of `Mclust` is almost the same as the original, whereas for `k-means` and `k-means++`, the cluster of Versicolor species slightly overlaps the cluster of Virginica. Regarding PCA, the two principal components in the plot explains almost 98% of the total variation of the dataset, meaning that it is sufficient to only consider these two components.