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 data points $\mathcal{X}$
ii. Choose the next center c_i by selecting $x' \in \mathcal{X}$ with weighted probability $\frac{D(x')^2}{\sum_{x \in \mathcal{X}} D(x)^2}$
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 calculations 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

For the `k-means` problem (thus `k-means++` as well), we wish to minimize

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

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

We now use the value of ϕ and the execution time of both algorithms to compare the perfomance 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	7.7130737×10^7	4.9900808×10^4	2.3567225×10^7	4.9900808×10^4	0.0071272	3.2105837
25	1.4935891×10^7	4.1182389×10^4	4.0574067×10^4	4.0607824×10^4	0.0206876	14.4335157
50	3.3113462×10^4	3.2406111×10^4	3.2253683×10^4	3.199007×10^4	0.0310799	49.819032

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

Exercise 2: `iris` dataset

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

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

```
#kmpp <- kmeanspp(data_iris, 3)
#kmpp_class <- kmpp$cluster
#table(kmpp_class, real_class)
```

```
#km <- kmeans(data_iris, centers=3)
#km_class <- km$cluster
#table(km_class, real_class)
```

```
#library(mclust)
#m <- Mclust(data_iris, G=3)
#m_class <- m$classification
#table(m_class, real_class)
```

2. Visualize the different partitions on PCA plan

```
#PCA(data_iris[kmpp_class==1,])
#PCA(data_iris[kmpp_class==2,])
#PCA(data_iris[kmpp_class==3,])
```