k-means 法の初期値

２０１６．０９．０２　応用数理研究所　　佐々木　俊久

クラスタリング手法は，データを自動的にグループ化する一種の教師なし学習手法である。階層型クラスタリングと非階層型クラスタリングがある。非階層型クラスタリングとして，k-means 法が多く用いられている。

１．k-means 法　－　特定のクラスター数に分類する非階層的手法

以下では，n 個の個体が p 変量データ $X_j = (x_{ij}, x_{2j}, \cdots, x_{pj}), (j = 1,2,\cdots,n)$ として与えられているとする。またクラスター数 k も与えられてものとする。

１．初期条件として，クラスター数 k 個のシード(中心)点，すなわちクラスターを形成する核となる個体を定める。

２．各個体について，すべてのシード点との距離を計算し，その距離が最も小さいシード点に対応するクラスターに入れる。

３．各クラスターの重心（平均）を求め，新たなシード点とする。

４．クラスターに変化が起こらないか，または，繰り返し回数に上限を設けるなど，あらかじめ設定された条件を満たした場合には終了し，それ以外のときは，ステップ２．からの手順を繰り返す。

以上のアルゴリズムは，クラスター内におけるデータ点のちらばりの度合いを表す**クラスター内平方和**[1]が小さくなるクラスターを求めることになる。

このアルゴリズムでは，初期条件のとり方によって最終結果が異なる可能性があり，クラスター内平方和が最小値となる保証はない。そのために，初期条件を変えて何回か実行するのが望ましい。

２．階層型手法のデンドログラムを利用する方法

階層型手法のデンドログラムを利用し，クラスター数 k 個に分ける。つまり適当な高さ(height)でデンドログラムのクラスター木を切って，各個体の属するグループを定め，各グループ内の個体の平均値をシード(中心)点とする。この方法だと常に k-means 法の結果は同じである。この方法は，S-PLUS[2]で採用されている方法である。

図２－１の例ですと，高さを 25 前後でグループ分けすればクラスター数を７つにすることができます。橙の線で切られた木の下が各クラスターの個体です。

1　S-PLUS によるデータマイニング入門　4.5.2 k-means 法について

2　NTT データ数理システムで販売している汎用統計解析ソフトウェア
kmeans，hclust，cutree **コマンド**

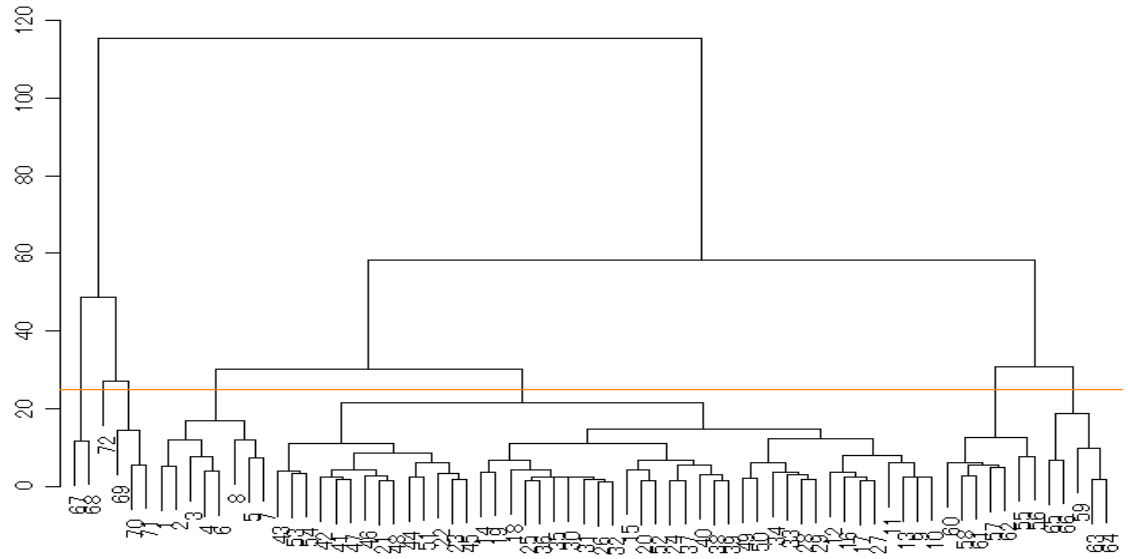

図２－１　クラスター数７の場合の初期条件の設定の高さ

３．ランダムに選択する方法

クラスタリング結果がランダムに選択された初期状態に依存するので，指定回数の複数回初期値を変えて，kmeans 計算を行い，クラスタ内平方和が最小となる初期値の結果を採用する。フリーな統計パッケージ R で採用している方法です。

S-PLUS で実現するコード

```
kmeans.R <- function(x, k, iter.max = 10, nstart = 1)
{
        n <- nrow(x)
        # number of data points
        centers <- numeric(k)
        # IDs of centers
        res.best <- list(tot.withinss = Inf)
        # the best result among <nstart> iterations
        for(rep in 1:nstart) {
                centers <- sample(1:n, k)
                ## Perform k-means with the obtained centers
                res <- kmeans(x, x[centers,  ], iter.max = iter.max)
                res$inicial.centers <- x[centers,  ]
                ## Store the best result
                if(sum(res$withinss) < res.best$tot.withinss) {
                        res.best <- res
```

```
                    res.best$tot.withinss <- sum(res$withinss)
                }
        }
        res.best
}
```

４．KKZ 法[3]

KKZ 法は，Katsavounidis ら(1994) によって提案され，初期のクラスタ中心として，最も離れているデータ同士をクラスタ中心の初期値として選択する手法である。

1) 与えられた全データから，データ同士の距離が最大となる 2 つのデータをクラスタ中心点に設定する。
2) 全データに対して，「既に決定されたクラスタ中心点とデータの最短距離」を求める。
3) 2) の値が最大となるデータを新たなクラスタ中心点とする。
4) 2) と 3) を必要数繰り返す。
5) 初期のクラスタ中心点がすべて決定したら，後は普通の k-means 。

この手法は，入力順序に依存せず，クラスタ同士の距離が離れたクラスタリング結果を得ることができるが，外れ値に敏感であるという問題がある.

R(S-PLUS)で，初期値（データの行番号）を求めるコード

```
kkz.init <- function(x, k)
{
      n <- dim(x)[1.]
      full <- matrix(0., n, n)
      full[lower.tri(full)] <- dist(as.matrix(x))
      dist1 <- full + t(full)
      ij <- which(dist1 == max(dist1))[1]
      i <- (ij-1)%%n + 1
      j <- (ij-1)%/%n + 1
      center <- rep(0, k)
      center[1] <- j
      min.dist <- dist1[j,]
```

[3] k-means 法の様々な初期設定によるクラスタリング結果の実験的比較

```
        center[2] <- i
        if( k > 2 ) {
                ipt <- i
                for(icl in c(3:k)) {
                        min.dist <- pmin(min.dist, dist1[ipt,])
                        ipt <- which(min.dist == max(min.dist))[1]
                        center[icl] <- ipt
                }
        }
        center
}
```

注)　R(S-PLUS)の dist 関数を使用しているのが，ビックデータでは気になります。

５．k-means++法

k-means++法は，David Arthur(2007)によって提案された。KKZ法の外れ値に弱いといった特性を改良しています。上記 3)で最大となるデータを選択したが，k-means++法では，必ずしも最大でなく，2) の距離の二乗を確率にしてルーレット選択による。

1) 与えられた全データからランダムに 1点を決め，それを 1つ目のクラスタ中心点にする。

2) 全データに対して,「既に決定されたクラスタ中心点とデータの最短距離」を求める。(d とする。)

3) 2) の値 d をそれぞれ 2乗する。(pow_d とする。)

4) 全データの pow_d をそれぞれ合計した値 (sum_pow_d) を用いて，各データの割合 (pow_d / sum_pow_d) を求める。

5) 4) で求めた割合を用いてルーレット選択を行い，新たなクラスタ中心点とする。

6) 2) ～ 5) を必要数繰り返す。

7) 初期のクラスタ中心点がすべて決定したら，後は普通の k-means。

k-means 法は，最初のデータ点をランダムに選択したり，ルーレット選択もしますので，初期値依存でもありますし，毎回結果が異なります。３章のように複数回行って，最良の結果を得る方法が，杉山 麿人[4]氏により R で実装するコードが公開されています。「R に付属の関数 kmeans を使って，K-means++をなるべく高速に実装します．特に必要となるライブラリはありません.」とのことです。

[4] http://mahito.info/kmeansp2.html　Mahito Sugiyama 2014

S-PLUS でのコード

```
> kmeansp2 <- function(x, k, iter.max = 10, nstart = 1)
{
        n <- nrow(x)
        # number of data points
        centers <- numeric(k)
        # IDs of centers
        distances <- matrix(numeric(n * (k - 1)), ncol = k - 1)
        # distances[i, j]: The distance between x[i,] and x[centers[j],]
        res.best <- list(tot.withinss = Inf)
        # the best result among <nstart> iterations
        for(rep in 1:nstart) {
                pr <- rep(1, n)
                # probability for sampling centers
                for(i in 1:(k - 1)) {
                        centers[i] <- sample(1:n, 1, prob = pr)
                        # Pick up the ith center
                        distances[, i] <- apply((t(x) - x[centers[i],  ])^
                                2, 2, sum)
                        # Compute (the square of) distances to the center
                        max.col <- apply( - distances[, 1:i, drop = FALSE],
                                1, function(x)  which(x == max(x))[1])
                        pr <- distances[cbind(1:n, max.col)]
                }
                centers[k] <- sample(1:n, 1, prob = pr)
                ## Perform k-means with the obtained centers
                res <- kmeans(x, x[centers,  ], iter.max = iter.max)
                res$inicial.centers <- x[centers,  ]
                ## Store the best result
                if(sum(res$withinss) < res.best$tot.withinss) {
                        res.best <- res
                        res.best$tot.withinss <- sum(res$withinss)
                }
        }
        res.best
}
```

付録　ルーレット選択

ルーレット選択はそれぞれに比重がある複数項目のうちからランダムに一つを選び出す方法です。R(S-PLUS)では，sample 関数の prob 引数でそれぞれの比重を指定すれば簡単に求まります。C(C++)等で乱数から計算するには，以下のような手順で求めます。

項目数をn,比重を$p_i\ (i=1,\ldots,n)$とする。

1) $sumP=\sum_{i=1}^{n}P_i$,

2) 次式を満たす実数値Lをランダムに決める。

$$0\le L\le s\ u\ m$$

3) Lに従って次式を満たすlを求める。

$$\sum_{i=1}^{l-1}Pi\le L\le\sum_{i=1}^{l}P_i$$

このl が選択された項目番号です。

＊ sample 関数での検証例

p1=0.2, p2=0.3, p3=0.5 での１００回のルーレット選択の結果

```
res<-rep(0,100)
for(i in 1:100)
  res[i] <- sample(1:3, 1, prob=c(0.2,0.3,0.5))
table(res)
1  2  3
23 29 48
```

p1=0.5, p2=0.3, p3=0.2 での１００回のルーレット選択の結果

```
res<-rep(0,100)
for(i in 1:100)
  res[i] <- sample(1:3, 1, prob=c(0.5,0.3,0.2))
table(res)
  1  2  3
  42 36 22
```

＊ 実数値Lを使用した方法での検証例

R(S-PLUS)でシミュレーション

```
cal.p  <- function(pr)
{
        pr1 <- c(0, cumsum(pr))
        maxL <- sum(pr)
        L <- runif(1, 0, maxL)
        num <- sum(pr1 < L)
```

```
        num
}
```

p1=0.2, p2=0.3, p3=0.5 での１００回のルーレット選択の結果

```
  res<-rep(0,100)
  for(i in 1:100)
    res[i] <- cal.p(c(0.2,0.3,0.5))
  table(res)
    1  2  3
   21 34 45
```

p1=0.5, p2=0.3, p3=0.2 での１００回のルーレット選択の結果

```
  res<-rep(0,100)
  for(i in 1:100)
    res[i] <- cal.p(c(0.5,0.3,0.2))
  table(res)
    1  2  3
   48 34 18
```

以上