第10回 k-means法

Code

1. 概要

k-means法は、データを複数個のクラスターに分割する方法の一つです。ここでは、k-means法のデモ・問題設定と仕組み・長所と注意点についてそれぞれ説明します。

2. k-means法のデモ

各国の乳製品の消費量のデータを持ちいて、消費量による国のクラスタリングをやってみましょう。

Hide

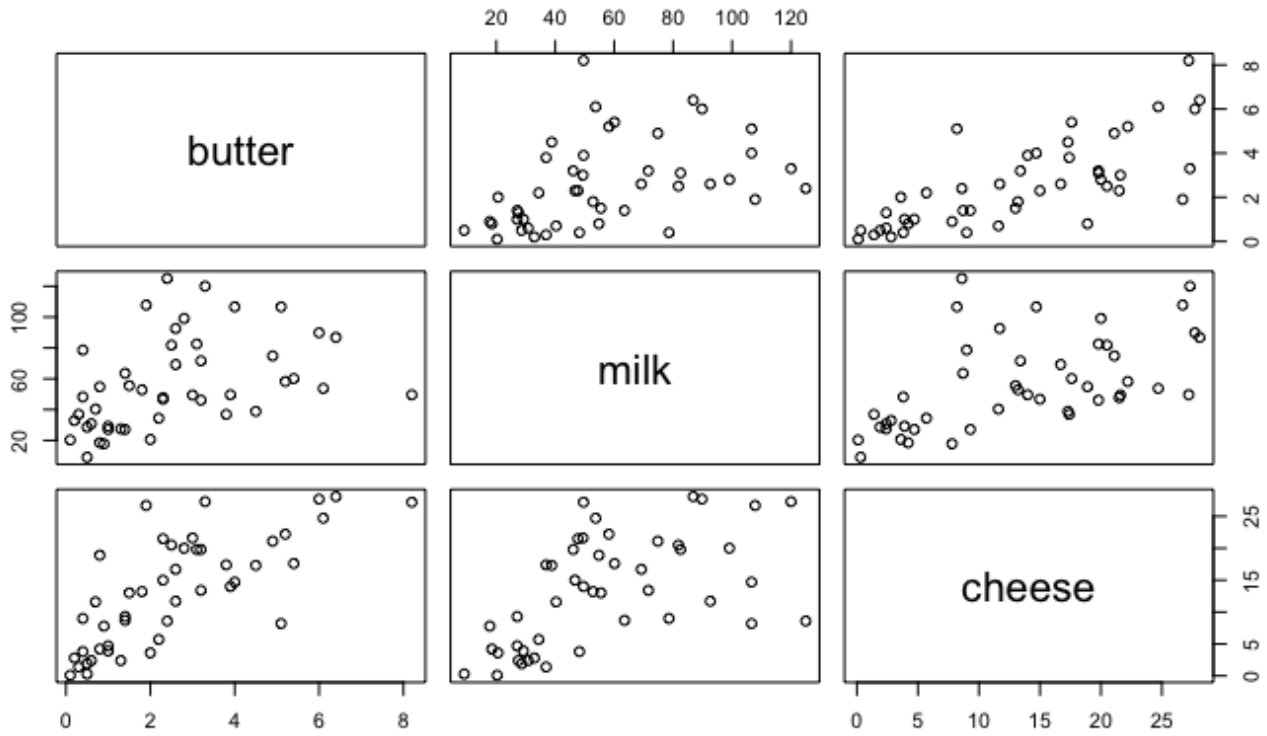
```
dat <- read.csv("./data/dairy_products.csv",
                fileEncoding = "utf-8",
                row.names = "country")
head(dat, n = 5)
```

	butter <dbl>	milk <dbl>	cheese <dbl>
中国	0.1	20.3	0.1
日本	0.6	30.8	2.4
韓国	0.2	32.9	2.8
イラン	1.0	27.1	4.7
トルコ	0.9	17.8	7.8

5 rows

Hide

```
plot(dat)
```



k-means法を行う際には、事前に正規化をおこないます。理由は5節で説明します。今回は正規化として標準化を選択しました。

Hide

```
dat_scaled <- scale(dat)
```

k-means法は、 kmeans 関数に以下の値を指定することで計算できます。

- x ：データ
- centers ：クラスター数
- iter.max ：繰り返しの最大数

今回は、4つのクラスターに分割してみます。

Hide

```
# k-means法の計算
result <- kmeans(x = dat_scaled, centers = 3, iter.max = 100)
result$centers
```

```
     butter       milk     cheese
1 -0.8046707 -0.68201079 -0.7930154
2  1.1583954  0.01339385  0.9687856
3  0.2403286  1.34819249  0.4411023
```

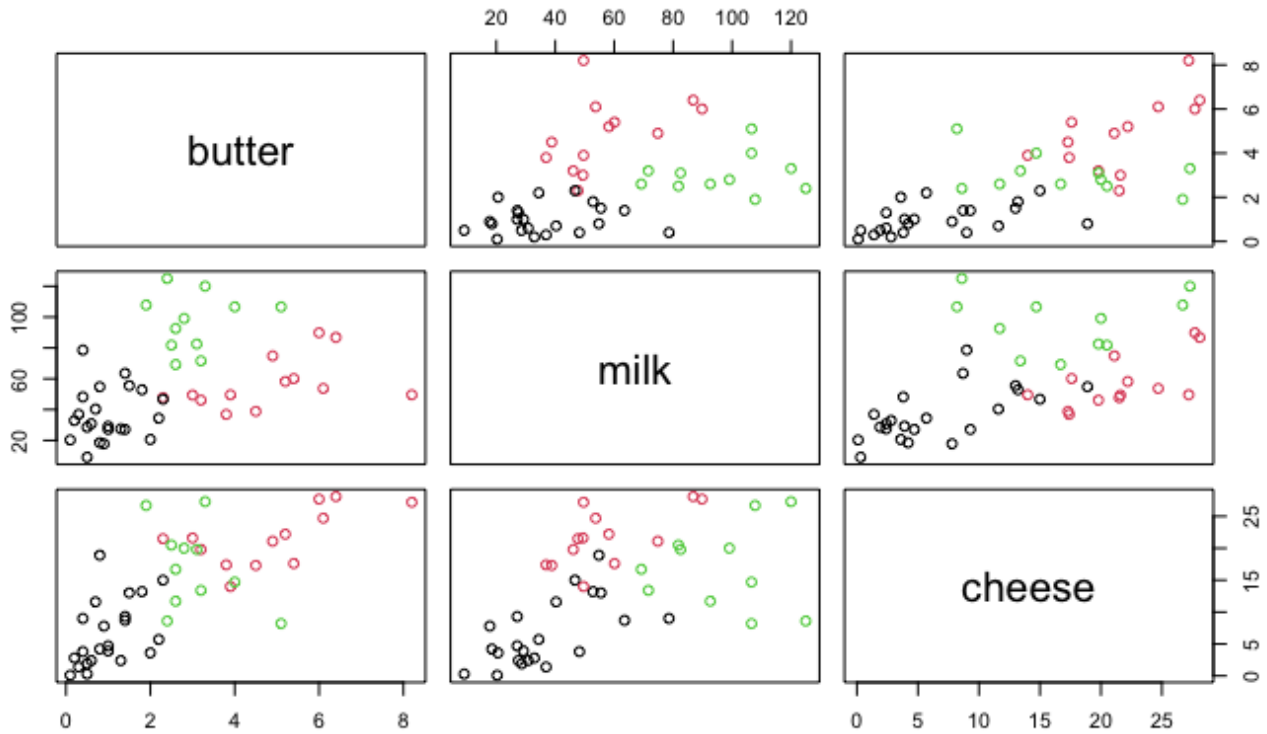
k-means法を行うと、それぞれのクラスターの中心と各データ点がどのクラスターに所属するかを求めることができます。クラスター中心は、得られたクラスターを解釈するヒントになります。今回の場合、

cluster 1 : 乳製品をあまり消費しない国 cluster 2 : butter, cheeseよりmilkでしょ！ cluster 3 : milkよりbutter, cheeseでしょ！

と解釈できます。また、クラスターは次のように分布しています。

Hide

```
# クラスターの可視化
plot(dat, col = result$cluster)
```



3. k-means法の問題設定と仕組み

A. 問題設定

数式による問題設定の説明

k-means法は、1956年にSteinhausが次のような問題を考えたことに始まります。

問題 : K をある正の整数に決めてください。d個の変数で表現される N 個の点の集合

$$D = \{x_n = (x_{n1}, \cdots, x_{nd}) \mid n = 1, 2, \cdots, N\}$$

を、以下の関数が最小になるように k 個の交わりのない集合 $C_1, \cdots, C_K$ に分割するにはどうすれば良いでしょうか。

$$L(C_1, \cdots, C_k) = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - c_k\|_2^2$$

ここで、$\|v\|_2 = \sqrt{v_1^2 + \cdots + v_d^2}$（いわゆるユークリッド距離）、$c_k = \frac{1}{\#C_k} \sum_{x \in C_k} x$（集合$C_k$に属する点の重心）です。

術語 : $C_1, \cdots, C_K$ をクラスター、c_k をクラスター C_k のcentroidと言います。

数式の解釈

問題の本質は N 個のデータ点 $x_1, \cdots, x_N$ が手元にあった時、これを自分で予め決めた個数 K のクラスター $C_1, \cdots, C_K$ に分割したいというものです。さて、どのようなクラスターを作りたいかは関数 $L(C_1, \cdots, C_K)$ に表現されています。注目してほしいのは、

$$\sum_{x \in C_k} \|x - c_k\|_2^2$$

という式です。この式は、クラスター C_k に所属するデータ点がその重心からどれだけ離れているかを表している式に読めます。そのクラスターが小さく纏まっているほど、この値は小さくなることが期待されるでしょう。いわゆるクラスター C_k の「分散」です。要するに、この問題はクラスターの分散の合計が最も小さくなるように、与えられた N 個のデータ点を K 個のクラスターに分割しなさいという問題を考えていることがわかります。

B. 得られるもの

k-means法の課題設定を解いた結果得られるものとして、

- クラスター中心
- 各データ点が所属するクラスターのid

の2つがあります。クラスターを作ったらそのクラスターの意味を解釈することが大切ですが、クラスター中心はこの解釈のヒントになります。

C. 仕組み

k-means法の課題設定をコンピュータで正確に解くことは、とても時間のかかる作業だということが知られています。そこでLloyd(1957)とForgy(1965)は独立に、この問題の近似的な解を導く以下のようなアルゴリズムを提案しました。

【Forgy-Lloyd algorithm】

1. **クラスターの初期値** : 各データポイントを何らかの方法でクラスターに割り振る。
2. 以下の操作を繰り返す。
    - **centoroidの更新** : 各クラスターに所属する点の重心を計算する。
    - **クラスターの再配置** : データポイントそれぞれについて、最近傍のcentroidを探し、対応するクラスターに再配置する。

Remark : R言語では、Hartigan-Wong algorithmがデフォルトになっている。こちらのアルゴリズムで得られる解の集合の方が、Forgy-Lloyd algorithmで得られる解の集合より真に小さい（包含されている）ことが知られており、砕けた表現をするなら、より正確に解が得られやすいからである。なお、 algorithm 引数に Forgy か Lloyd を渡すことで、Forgy-Lloyd algorithmを用いることができる。 Forgy , Lloyd と区別されているが、どちらを選択しても全く同じアルゴリズムが動く。■

4. Forgy-Lloyd algorithmのデモ

試しにこのアルゴリズムによるクラスターの更新の様子を見てみましょう。まず、デモデータを次のように生成します。可視化した結果から確認できますが、今回は2次元の散布図上で2群（左下と右上）に分かれているデータを準備しました。

Hide

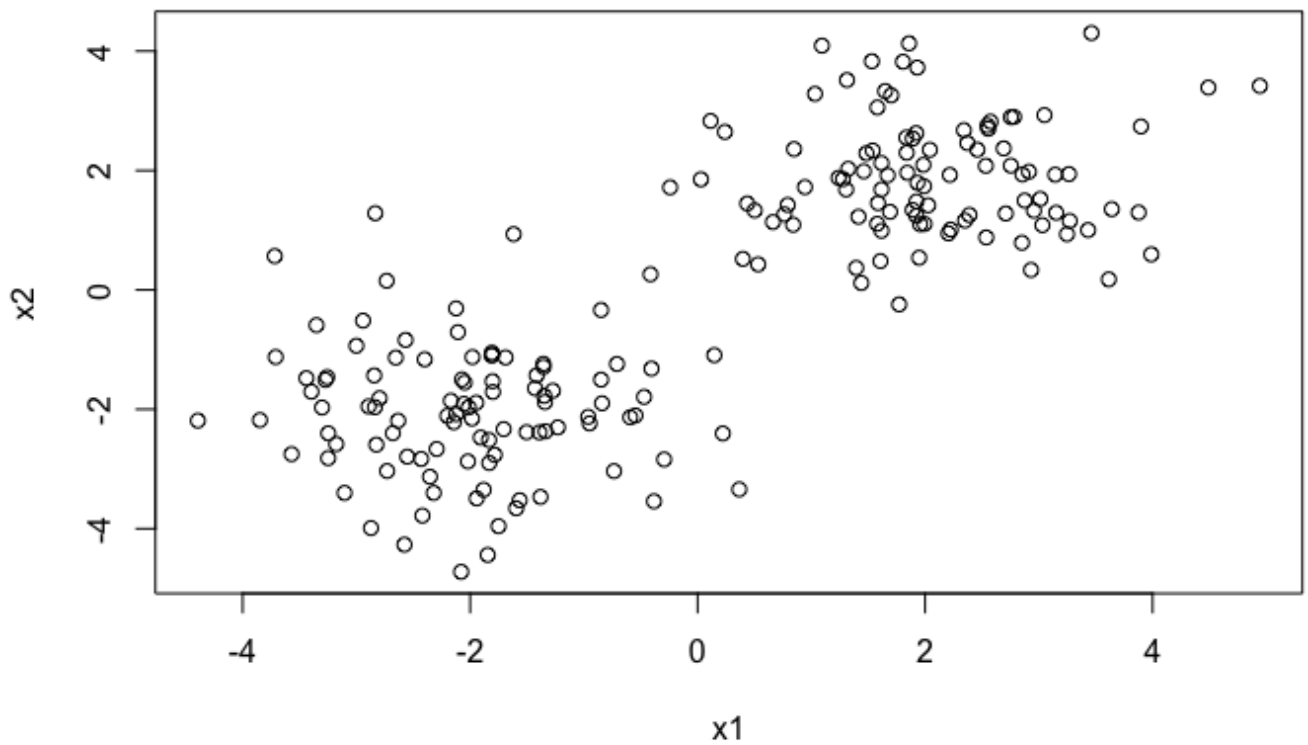
```
# デモデータの生成
library(MASS)
cluster_1 <- mvrnorm(n = 100, mu = c(-2, -2), Sigma = diag(c(1, 1)))
cluster_2 <- mvrnorm(n = 100, mu = c(2, 2), Sigma = diag(c(1, 1)))
X <- rbind(cluster_1, cluster_2)
dat <- data.frame(x1 = X[ , 1], x2 = X[, 2])
head(dat, n = 5)
```

	x1 <dbl>	x2 <dbl>
1	-1.3422282	-1.8800616
2	-2.9416343	-0.5146741
3	-2.8716507	-3.9900049
4	0.1466697	-1.0934696
5	-2.0191856	-2.8731611

5 rows

Hide

```
plot(dat)
```

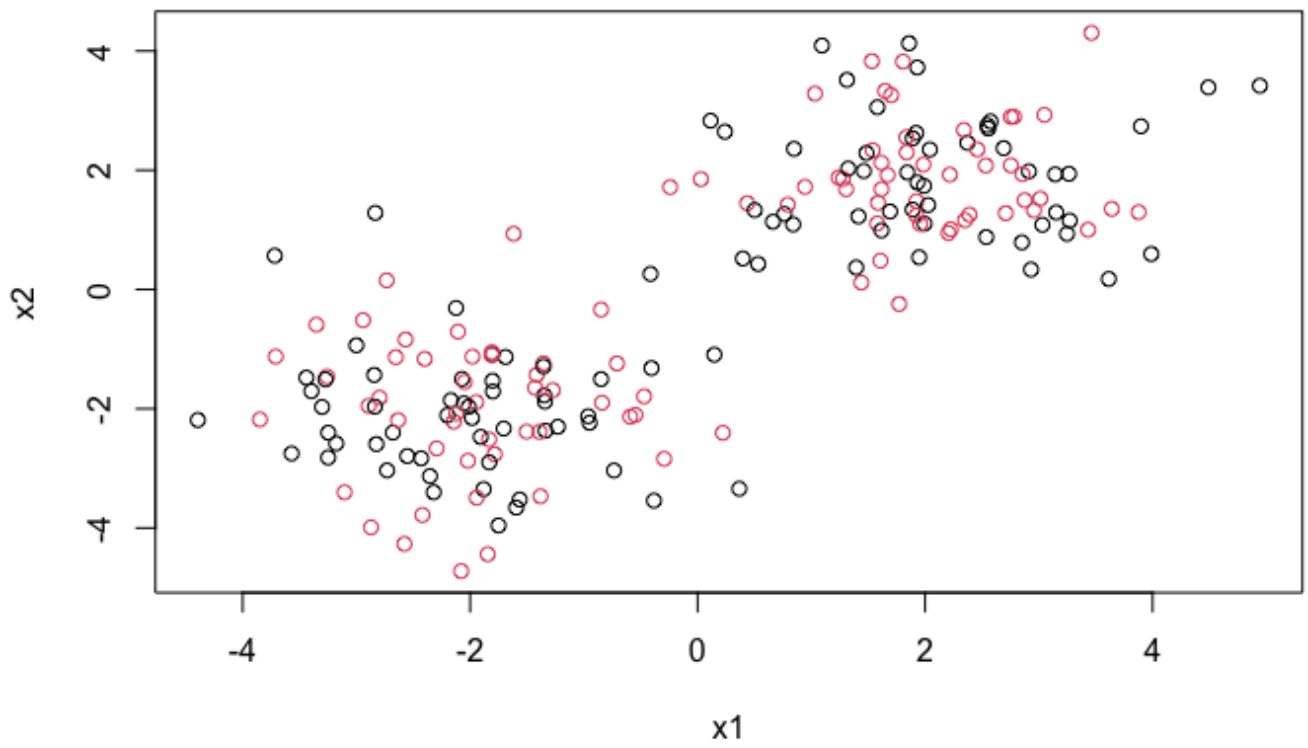


このデータを、クラスター数K=2のk-means法のアルゴリズムによってクラスタリングする様子を見ていきます。

step 1. クラスターの初期値 : 各データポイントを何らかの方法でクラスターに割り振る。今回は、各クラスターに対して等確率に割り振ることにします。

Hide

```
y <- sample(x = c(1, 2), size = 200, replace = TRUE)
plot(dat, col = y)
```



step 2. centroidの更新 : 各クラスターの重心を計算する。

Hide

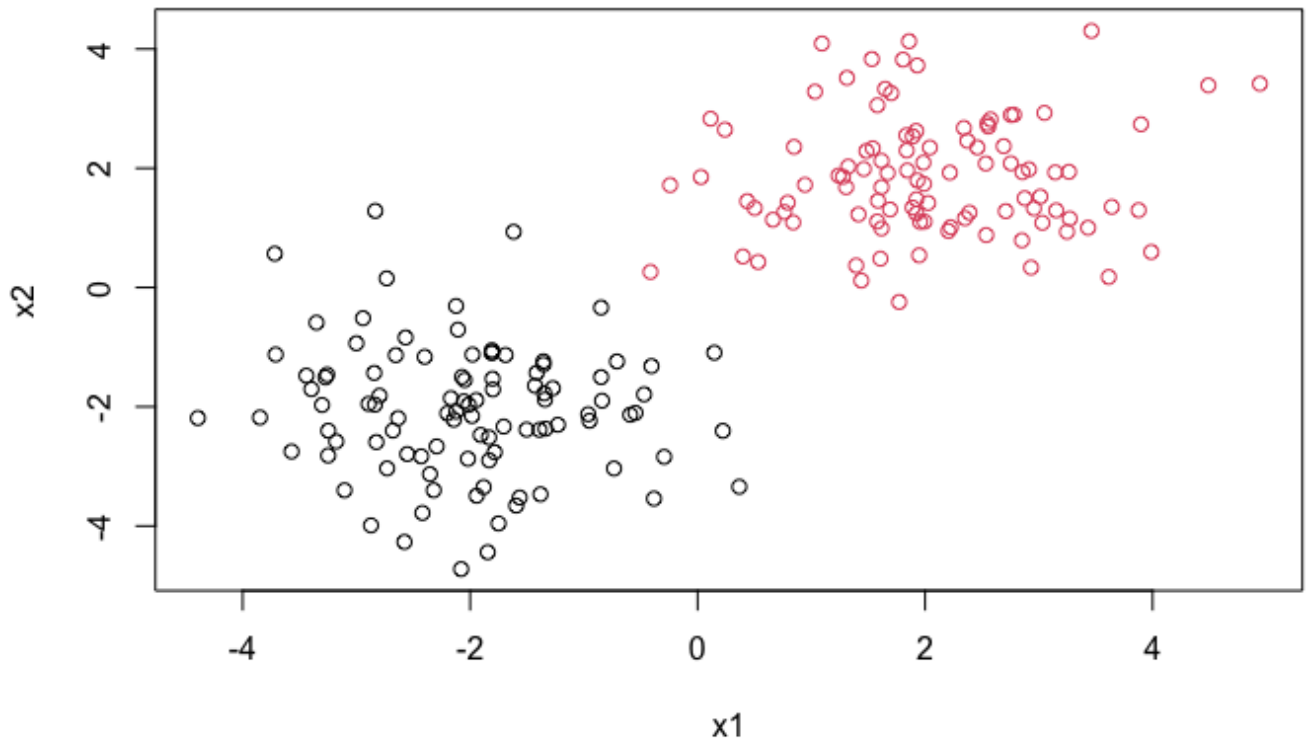
```
center_1 <- colMeans(dat[y==1, ])
center_2 <- colMeans(dat[y==2, ])
c(center_1, center_2)
```

```
        x1         x2         x1         x2 
-0.022538714 -0.130764046  0.065717687 -0.009468612 
```

step 3. クラスターの再配置 : データポイントそれぞれについて、最近傍のcentroidを探し、対応するクラスターに再配置する。

Hide

```
dist_1 <- rowSums(sweep(dat, MARGIN = 2, center_1)^2)
dist_2 <- rowSums(sweep(dat, MARGIN = 2, center_2)^2)
new_cluster_id <- 1*(dist_1 < dist_2) + 2*(dist_1 >= dist_2)
plot(dat, col = new_cluster_id)
```



5. クラスターの初期値の決め方

クラスターの初期値は、解の精度に影響を与えることが知られています。例えば、クラスターの初期値の決め方として最初に思いつくのは、各データ点をランダムにクラスターに割り振る方法でしょう。しかしこの初期値を採用した場合、Forgy-Lloyd algorithmによって得られる解は、関数

$$L(C_1, \cdots, C_K) = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - c_k\|_2^2$$

の値が真の最適な分割を与えるクラスターによる値よりいくらでも悪くなり得る例があることが知られています。以下のデータセットに対して、繰り返しk-means法を実行してみてください。

Hide

```
# デモデータの生成
cluster1 <- mvrnorm(n = 100, mu = c(6, 10), Sigma = diag(c(2, 2)))
cluster2 <- mvrnorm(n = 100, mu = c(13, 10), Sigma = diag(c(2, 2)))
cluster3 <- mvrnorm(n = 100, mu = c(10, -10), Sigma = diag(c(2, 2)))
cluster4 <- mvrnorm(n = 100, mu = c(-10, -10), Sigma = diag(c(2, 2)))
cluster5 <- mvrnorm(n = 50, mu = c(-10, 10), Sigma = diag(c(1, 1)))
X <- rbind(cluster1, cluster2, cluster3, cluster4, cluster5)
dat <- data.frame(x1 = X[ , 1], x2 = X[ , 2])
head(dat, n = 5)
```

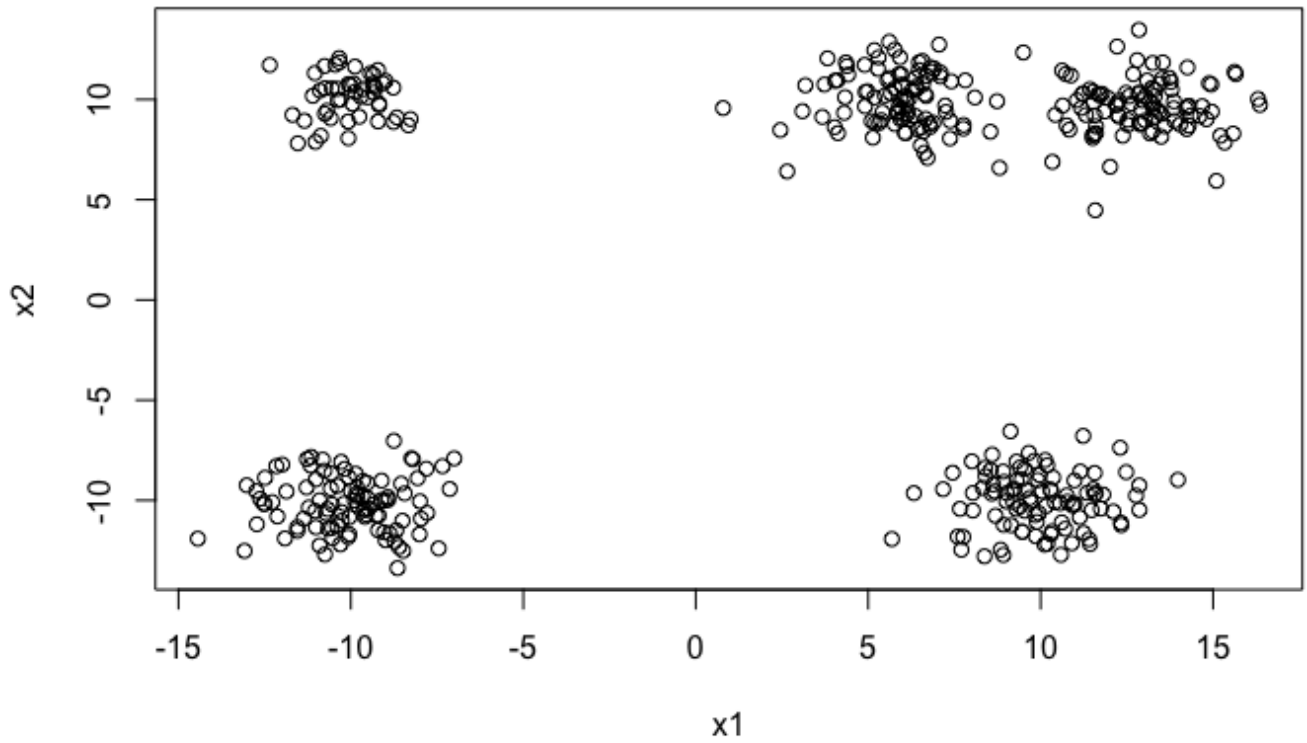
	x1 <dbl>	x2 <dbl>
1	5.610039	11.082842
2	4.908799	9.659212

	x1 <dbl>	x2 <dbl>
3	5.377743	8.966763
4	5.963044	9.576589
5	2.650137	6.411516

5 rows

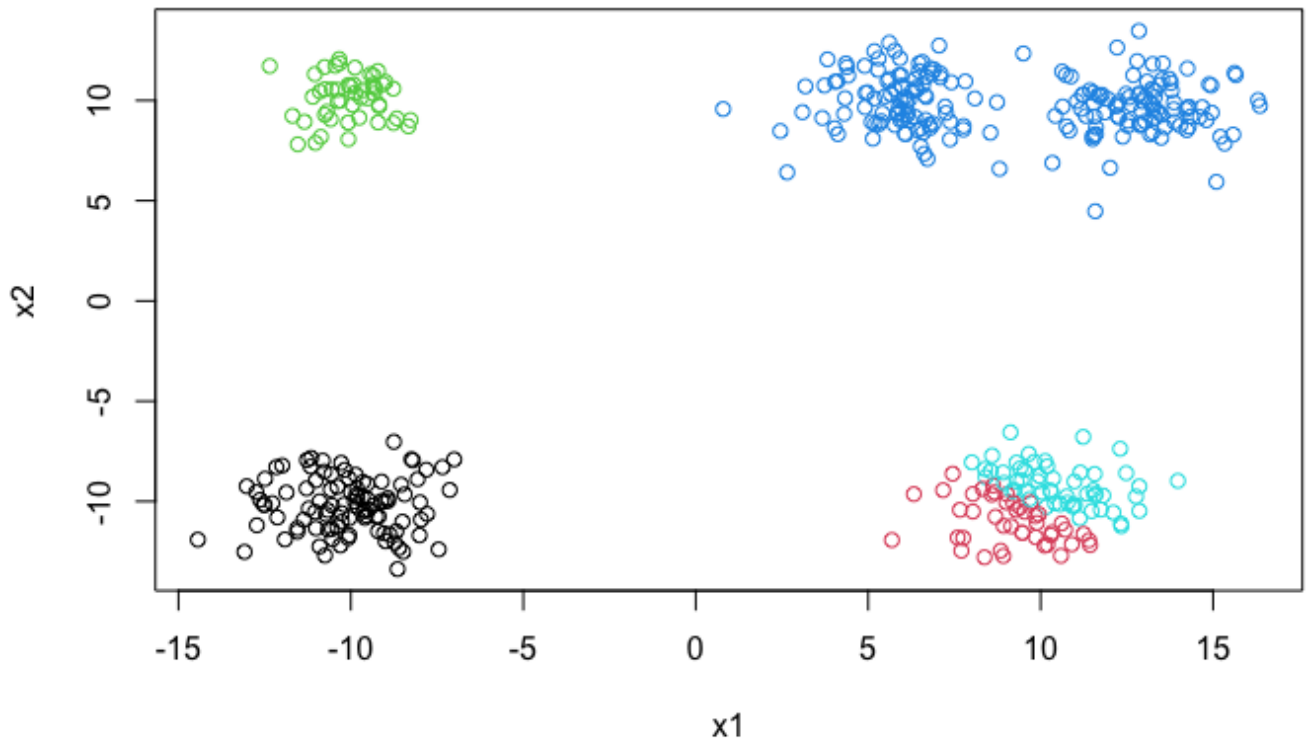
Hide

```
# 散布図 : 5つのクラスターが確認できる。
plot(dat)
```



Hide

```
# 20回くらい何度も実行して、結果を比較してみてください。
result <- kmeans(x = dat, centers = 5, iter.max = 100)
plot(dat, col = result$cluster)
```



そこで、2007年にArthurとVassilvistskiiは、この問題を緩和する初期値の決め方として**k-means++**を発表しました。

【k-means++ algorithm】

1. c_1 はデータ点の中から等確率でランダムに選ぶ。
2. 以下の計算を繰り返す。（tはステップ数とする。）
    - 各データ点に対して、$D(x) := \min_{k=1,\cdots,t} \|x - c_k\|$ を計算する。
    - c_{t+1} を確率 $D(x)^2 / \sum_x D(x)^2$ でランダムに選ぶ。

定理 : 関数 $L(C_1, \cdots, C_K)$ の最小値を L_{opt} , k-means++を初期値に採用したForgy-Lloyd algorithmによって得られるクラスターが与える関数 L の値を L_{++} と書くことにしましょう。このとき、以下のような精度の評価が成り立ちます。

$$\mathbb{E}[L_{++}] \leq 8(\log K + 2)L_{opt}$$

k-means++を自分で実装してみることにしましょう。実装してみたら、先程のデモデータに対するクラスターの結果がどうなるかを確認します。

Hide

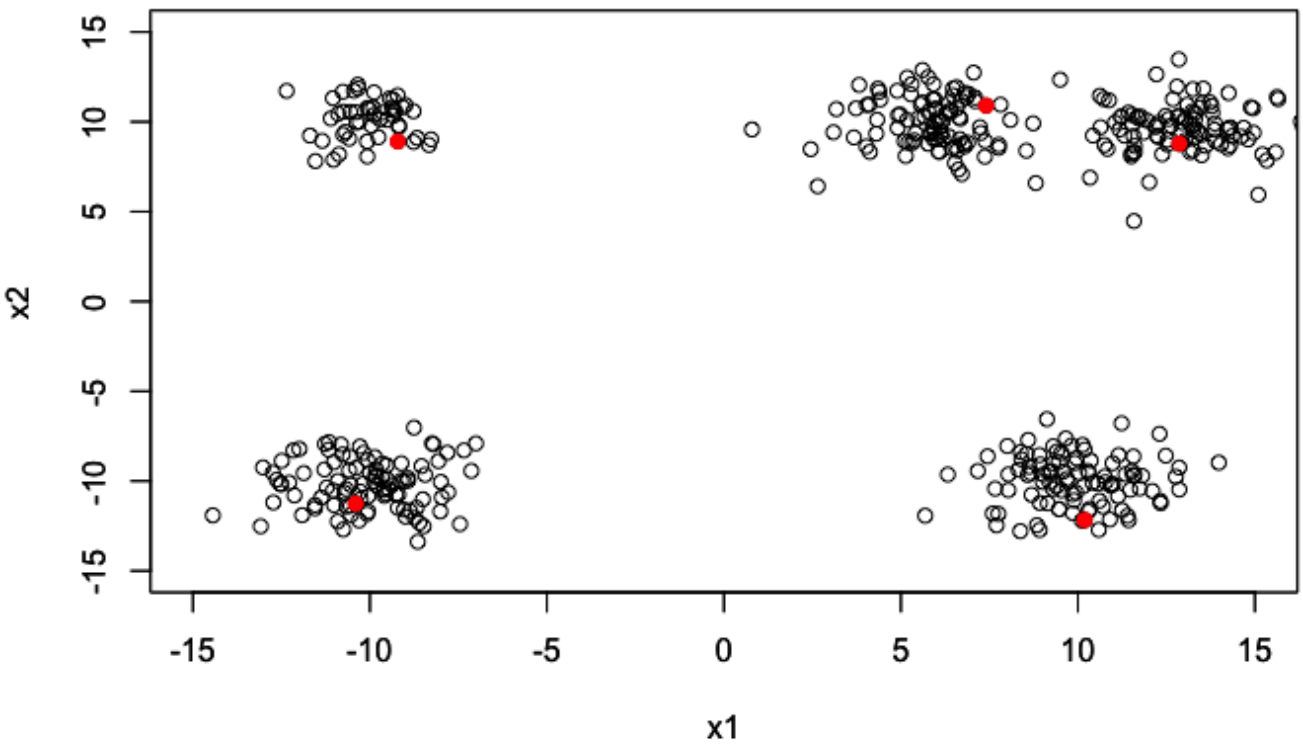
```
# kmeans++の実装
init_kmeanspp <- function(x, centers) {
  x <- as.matrix(x)
  dist_centers <- matrix(Inf, nrow = nrow(x), ncol = centers-1)

  center_pts <- sample(x = 1:nrow(x), size = 1)    # ランダムに一点選ぶ。

  for (i in 1:(centers-1)) {
    dist_centers[ , i] <- rowSums(sweep(x, 2, x[center_pts[i], ])^2)
    min_dist <- apply(dist_centers, MARGIN = 1, FUN = min)
    prob <- min_dist / sum(min_dist)
    center_pts[i+1] <- sample(x = 1:nrow(x), size = 1, prob = prob)
  }

  return(center_pts)
}
```

Hide

```
# どんなクラスターの初期値が選ばれているかを確認する。（複数回）
initial_centers <- init_kmeanspp(x = dat, centers = 5)
plot(dat, xlim = c(-15, 15), ylim = c(-15, 15))
par(new = TRUE)
```

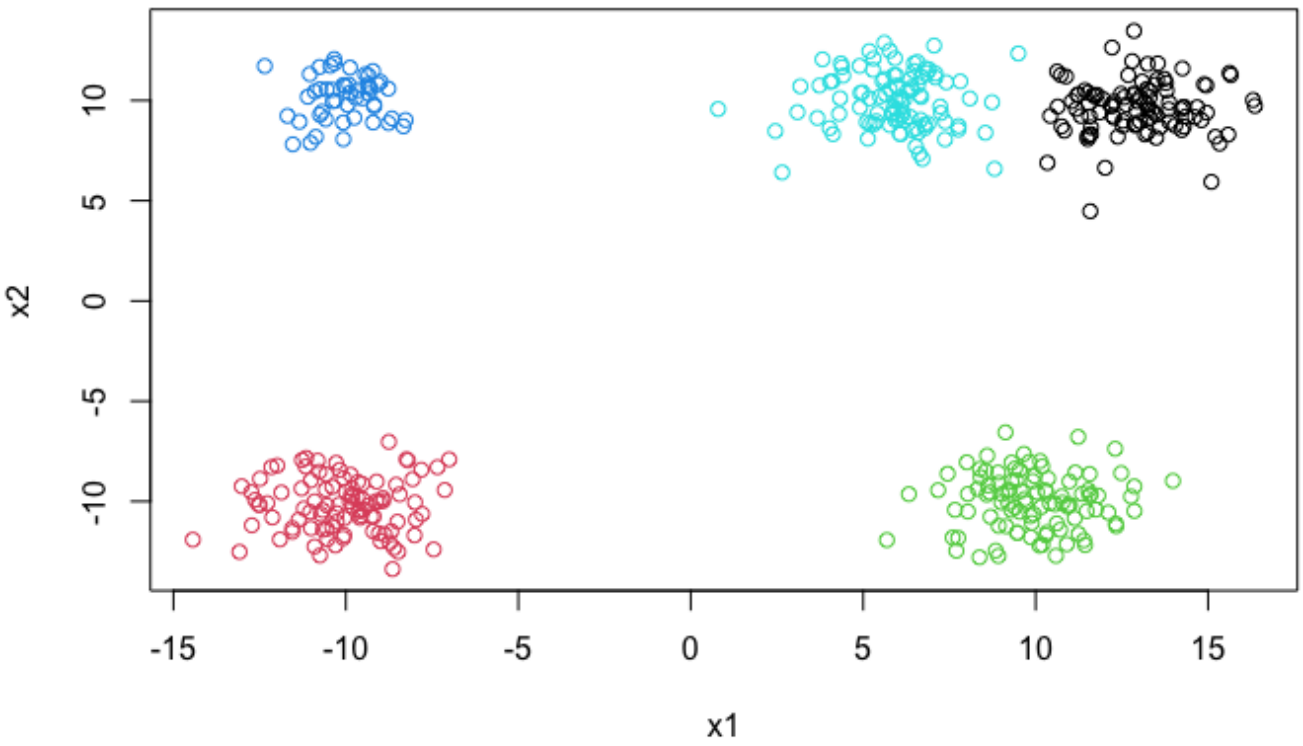
Hide

```
plot(dat[initial_centers, ], xlim = c(-15, 15), ylim = c(-15, 15),
     col = "red", pch = 19)
```



Hide

```
# 同様に20回くらい試してみてください。
initial_centers <- init_kmeanspp(x = dat, centers = 5)
result <- kmeans(dat, centers = dat[initial_centers, ], iter.max = 100)
plot(dat, col = result$cluster)
```



上記の実験を20回ほど試してみると、ランダムに初期値を決める場合より、k-means++のほうが適切なクラスターが行われやすい様子を確認できるのではないでしょうか。

5. k-means法の長所と注意点

k-means法は他のクラスタリング手法に比べて、データポイントの数に対してスケールする（計算量のオーダーが低い）という性質が知られています。また、アイディアの分かりやすさもあって、他のクラスタリング手法と比較して最初に試される手法です。

一方で、k-means法には2つの注意点があります。

1. データを正規化しておく必要性
2. 各クラスターのサンプルサイズの不均一性に対する弱さ

A. 正規化の必要性

k-means法のアルゴリズムは距離に依存しているので、ある変数の値が10倍されると、クラスタリングの結果がその変数に引っ張られてしまいます。実際に、以下のスクリプトを実行してみてください。

Hide

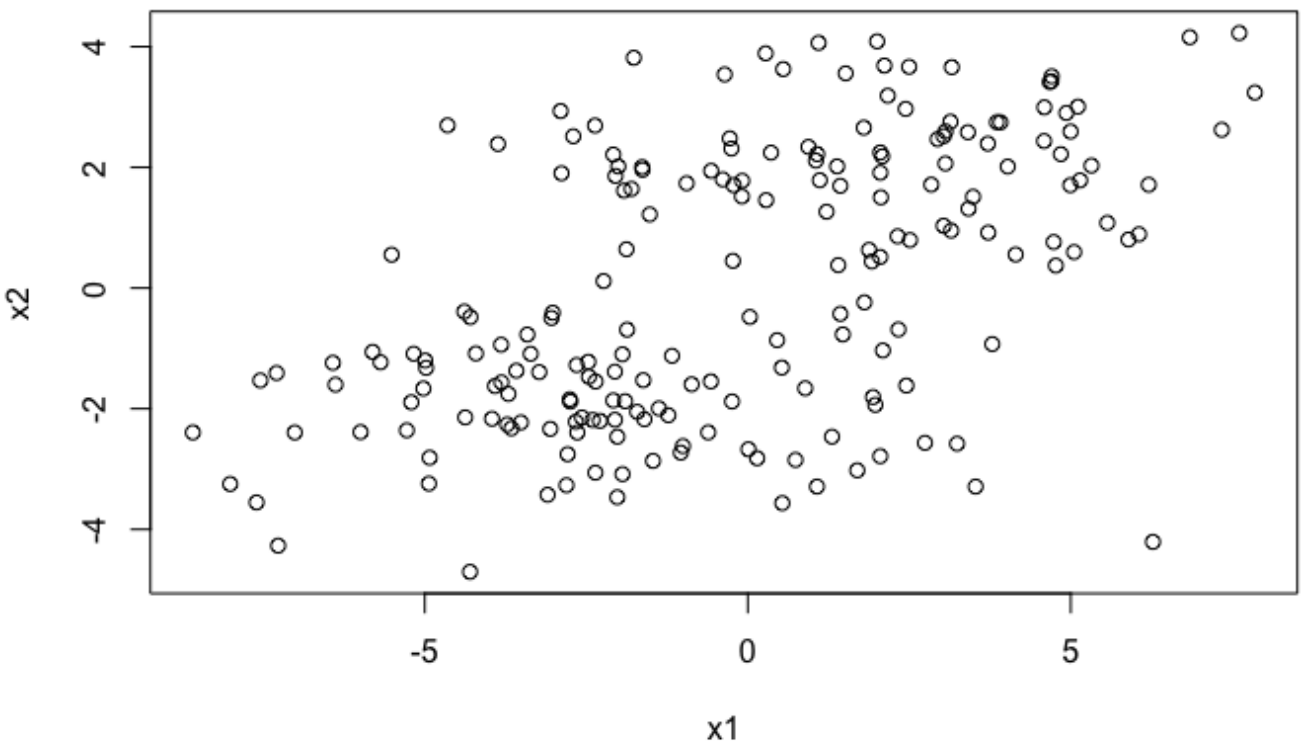
```
# デモデータの生成
cluster_1 <- mvrnorm(n = 100, mu = c(-2, -2), Sigma = diag(c(10, 1)))
cluster_2 <- mvrnorm(n = 100, mu = c(2, 2), Sigma = diag(c(10, 1)))
X <- rbind(cluster_1, cluster_2)
dat <- data.frame(x1 = X[ , 1], x2 = X[, 2])
head(dat, n = 5)
```

	x1 <dbl>	x2 <dbl>
1	-3.098442	-3.4257766
2	1.973203	-1.9363948
3	-1.867907	-0.6898445
4	-5.685633	-1.2301688
5	-7.604448	-3.5528874

5 rows

Hide

```
plot(dat)
```



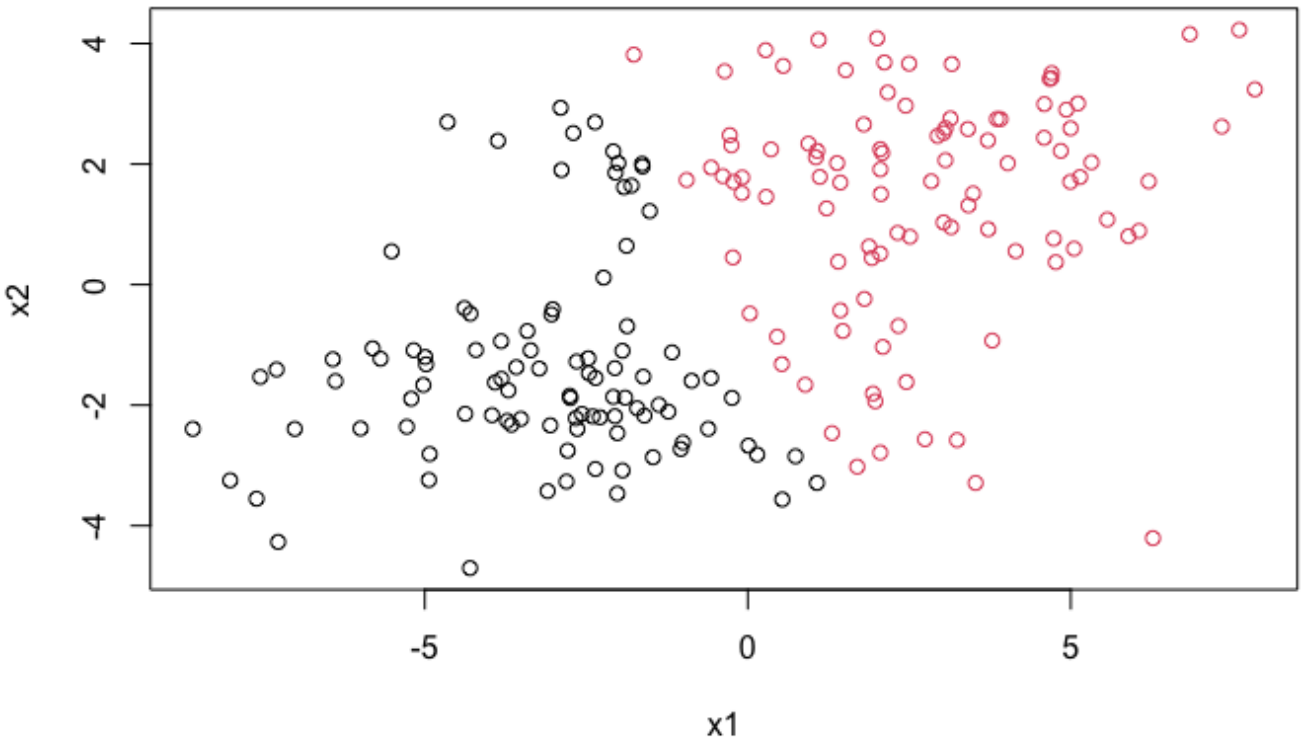
Hide

```
# k-means法によるクラスタリング
result <- kmeans(x = dat, centers = 2, iter.max = 100)
result$centers
```

```
        x1        x2
1 -3.142386 -1.366410
2  2.674283  1.400258
```

Hide

```
# クラスタリングの結果の可視化
plot(dat, col = result$cluster)
```



B. サンプルサイズの不均一性

また、k-means法は各データポイントのサンプルサイズがおおよそ同じであるという仮定を暗に置いています。実際、以下のスクリプトを実行してみてください。

Hide

```
# デモデータの生成
cluster_1 <- mvrnorm(n = 200, mu = c(-2, -2), Sigma = diag(c(1, 1)))
cluster_2 <- mvrnorm(n = 10, mu = c(2, 2), Sigma = diag(c(1, 1)))
cluster_3 <- mvrnorm(n = 20, mu = c(2, -2), Sigma = diag(c(1, 1)))
X <- rbind(cluster_1, cluster_2, cluster_3)
dat <- data.frame(x1 = X[ , 1], x2 = X[, 2])
head(dat, n = 5)
```

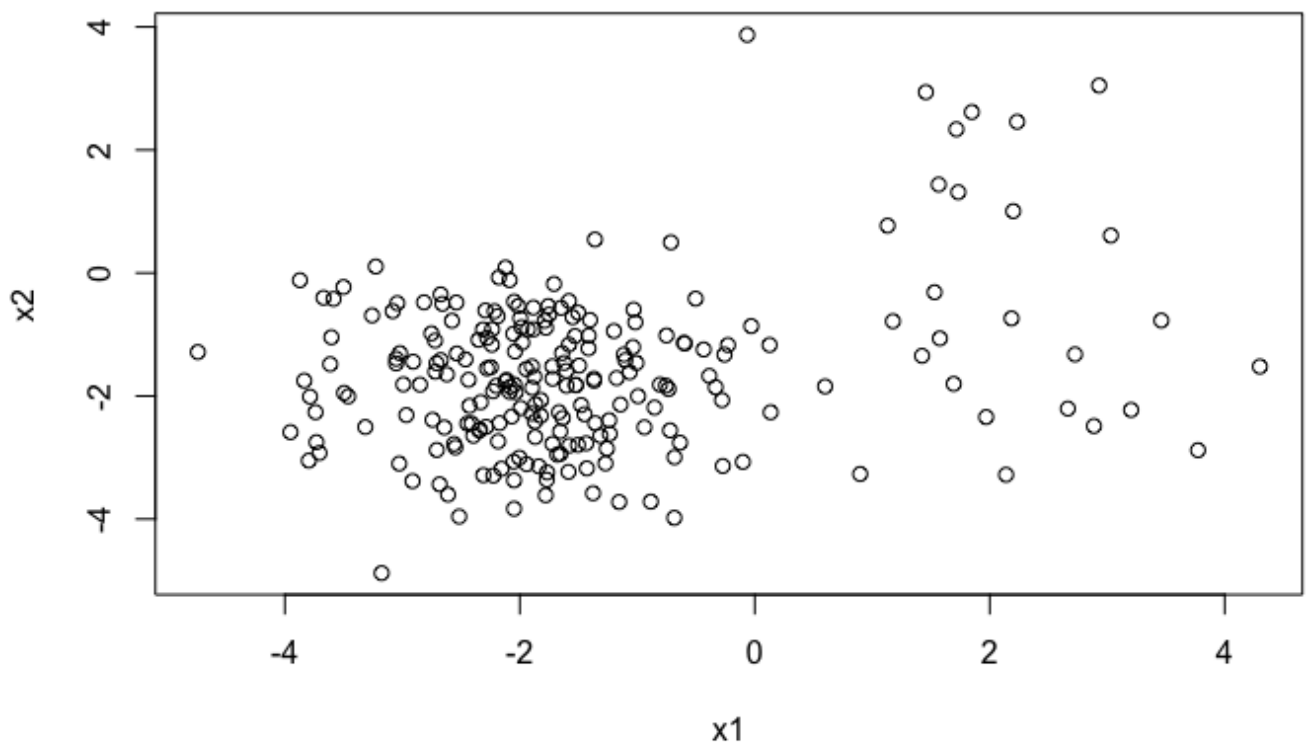
	x1 <dbl>	x2 <dbl>
1	-1.2590592	-2.85520688
2	-2.2474017	-1.53237654
3	-2.1214116	0.08793986

	x1 <dbl>	x2 <dbl>
4	-0.7209452	-2.55823774
5	-3.7387405	-2.26274485

5 rows

Hide

```
plot(dat)
```



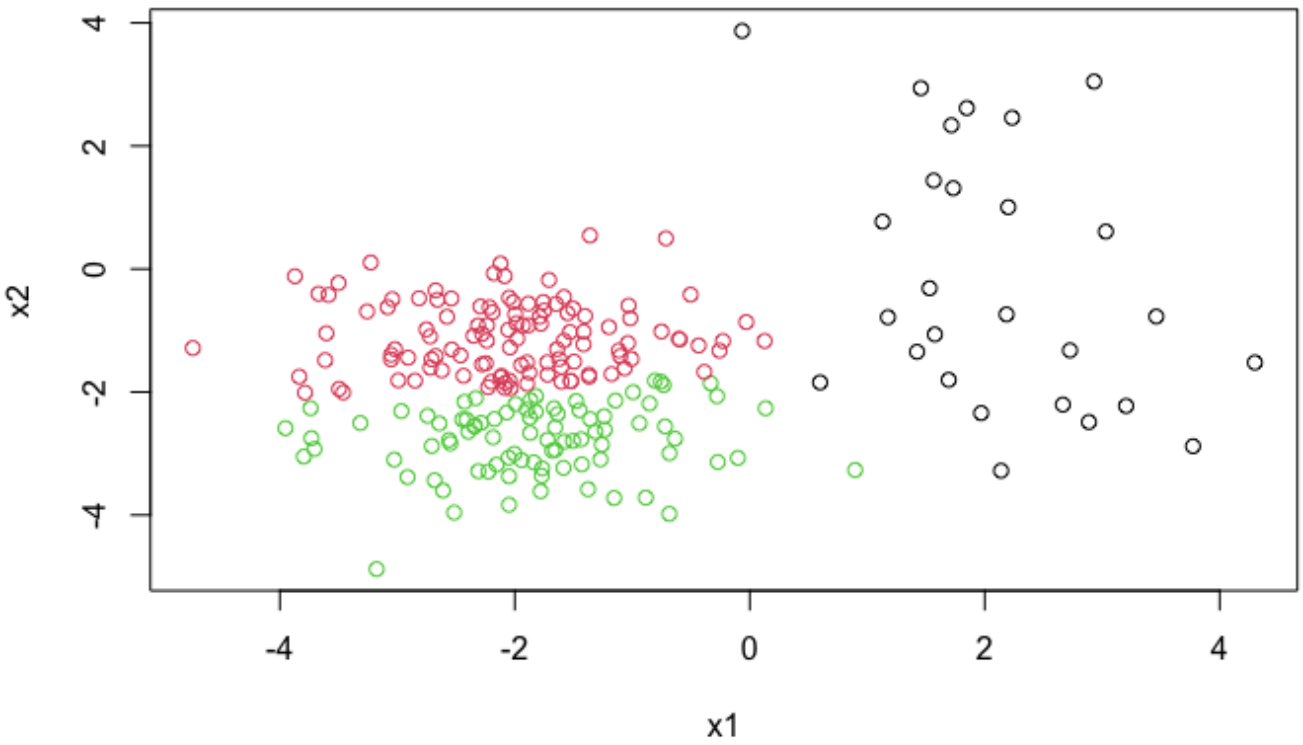
Hide

```
# k-means法によるクラスタリング
result <- kmeans(x = dat, centers = 3, iter.max = 100)
result$centers
```

```
        x1         x2
1  2.114349 -0.1677053
2 -2.042958 -1.1106878
3 -1.819332 -2.7713581
```

Hide

```
# クラスタリングの結果の可視化
plot(dat, col = result$cluster)
```


x2
4
2
0
-2
-4
-4
-2
0
2
4
x1