k-Means Clustern in R

Achim Zeileis

2009-02-20

Um die Ergebnisse aus der Vorlesung zu reproduzieren, wird zunächst wieder der `GSA` Datensatz geladen

```
R> load("GSA.rda")
```

und wie schon in den vorangegangenen Tutorien aggregiert. Die nach `country` aggregierten "Erfolgsanteile" für 8 verschiedene Sommeraktivitäten werden berechnet:

```
R> gsa <- GSA[, c(5, 10:12, 14, 25:27, 29)]
R> gsa <- na.omit(gsa)
R> sucrate <- function(x) prop.table(table(x))[2]
R> gsa <- aggregate(gsa, list(gsa$country), sucrate)
R> rownames(gsa) <- gsa[, 1]
R> gsa <- gsa[, -(1:2)]
```

und skaliert:

```
R> gsa <- scale(gsa)
```

Letzteres ist (wie schon in den vorangegangenen Analysen) notwendig, da die unterschiedlichen Sommeraktivitäten sehr unterschiedliche Popularität genießen.

Die skalierten Daten sollen nun mit Hilfe von k-Means geclustert werden. Da `kmeans()` per Voreinstellung die Ausgangsmittelwerte per Zufallsgenerator wählt, wird hier der sogenannte *random seed* (Ausgangswert für den Zufallszahlengenerator) gesetzt, damit die Ergebnisse dieses Tutoriums exakt reproduziert werden können.

```
R> set.seed(123)
```

Der Aufruf von `kmeans()` ist sehr einfach: man übergibt einfach die Daten und die gewünschte Anzahl k von Clustern. Alternativ koennte man auch konkrete Ausgangsmittelwerte angeben.

```
R> km2 <- kmeans(gsa, 2)
R> km2

K-means clustering with 2 clusters of sizes 10, 5

Cluster means:
  SA01.tennis SA02.cycle  SA03.ride  SA05.swim  SA17.shop SA18.concert
1  -0.4967063 -0.5656591 -0.5765824 -0.4067836  0.5456768    0.4160593
2   0.9934126  1.1313181  1.1531649  0.8135672 -1.0913537   -0.8321186
  SA19.sight SA21.museum
1  0.4061163   0.2847266
```

```
2 -0.8122326  -0.5694531

Clustering vector:
Austria (Vienna)  Austria (other)          Belgium          Denmark
               2                2                1                1
          France          Germany          Hungary            Italy
               1                2                2                1
     Netherlands            Spain           Sweden      Switzerland
               1                1                1                2
              UK              USA            other
               1                1                1

Within cluster sum of squares by cluster:
[1] 48.894450  9.661277

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

Das zurückgelieferte Objekt `km2` ist eine Liste mit verschiedenen Komponenten. `km2$cluster` enthält den Vektor mit Clusterzugehörigkeiten der Beobachtungen. `km2$centers` sind die k Mittelwerte der Cluster und `km2$withinss` gibt die Fehlerquadratsumme *WSS* innerhalb jedes Clusters an.

In diesem konkreten Fall entspricht Cluster 1 den kulturinteressierten Urlaubern: die sportlichen Aktivitäten finden nur unterdurchschnittliches Interesse (deutlich < 0), während die kulturellen Aktivitäten deutlich positives Interesse finden. Cluster 2 hingegen enthält die sportinteressierten Urlauber. Diese scheinen noch deutlicher gegen den ersten Cluster abgegrenzt zu sein: die Absolutwerte der skalierten Interessensvariablen sind noch größer, d.h. die Präferenz von Sport bzw. das Nicht-Interesse an Kultur ist noch ausgeprägter.

Um zu zeigen, daß der Algorithmus nur ein lokales Minimum findet, führen wir denselben `kmeans`-Aufruf noch einige Male durch und vergleichen jedesmal die assoziierte Fehlerquadratsumme *RSS*:

```
R> sum(km2$withinss)

[1] 58.55573

R> sum(kmeans(gsa, 2)$withinss)

[1] 58.55573

R> sum(kmeans(gsa, 2)$withinss)

[1] 58.55573

R> sum(kmeans(gsa, 2)$withinss)

[1] 57.74083

R> sum(kmeans(gsa, 2)$withinss)

[1] 58.55573
```

Um die Chance in einem lokalen Minimun hängen zu bleiben zu verringern, kann man den Algorithmus einfach 20 Mal durchführen und nur das Ergebnis behalten, das die geringste *RSS* hat. Dies kann man entweder 'von Hand' machen oder indem man das Argument `nstart` setzt:

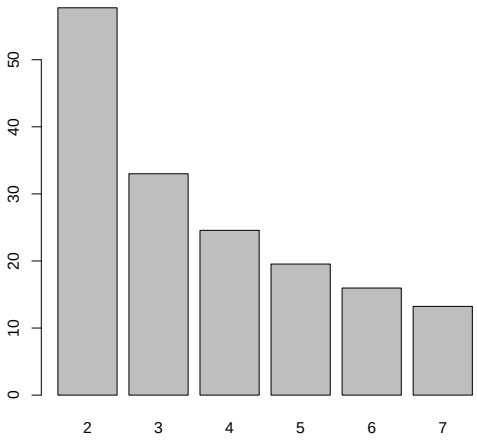
```
R> km2 <- kmeans(gsa, 2, nstart = 20)
```

Völlig analog berechnen wir uns das beste Ergebnis aus 20 Läufen fuer $k = 3, \ldots, 7$.

```
R> km3 <- kmeans(gsa, 3, nstart = 20)
R> km4 <- kmeans(gsa, 4, nstart = 20)
R> km5 <- kmeans(gsa, 5, nstart = 20)
R> km6 <- kmeans(gsa, 6, nstart = 20)
R> km7 <- kmeans(gsa, 7, nstart = 20)
```

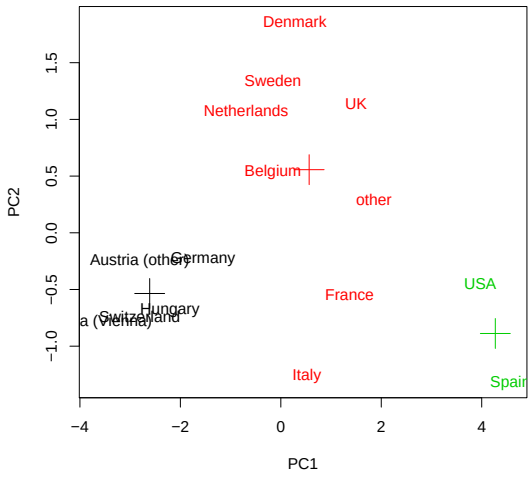
Um die Anzahl k von Clustern zu wählen, schaut man sich den Verlauf der *RSS* für eine wachsende Zahl k an. Dort, wo diese Kurve einen 'Knick' hat, also die Fehlerquadratsumme mit einem weiteren Cluster nicht mehr entscheidend sinkt, hört man in aller Regel auf. Die folgenden Befehle sammeln zunächst die *RSS* der verschiedenen Objekte ein und erzeugt dann einen `barplot`.

```
R> wss <- c(sum(km2$withinss), sum(km3$withinss), sum(km4$withinss),
+      sum(km5$withinss), sum(km6$withinss), sum(km7$withinss))
R> names(wss) <- 2:7
R> barplot(wss)
```



Dem Barplot zufolge würde man mindestens 3 Cluster wählen, evtl. auch 4. Um die Ergebnisse für $k = 3$ und $k = 4$ zu vergleichen, visualisieren wir die Daten zunächst. Dafür zeichnen wir ein Streudiagramm der ersten beiden Hauptkomponenten der Daten und anstatt von Punkten verwenden wir wieder die Labels der Beobachtungen als Plot-Symbole. Zusätzlich färben wir die Labels noch nach den Cluster-Zugehörigkeiten `km3$cluster` ein. Um zu sehen, wo die Cluster-Mittelwerte liegen, werden auch diese mit Hilfe des Befehls `points` hinzugefügt. Dabei ist zu beachten, daß die Klassenmittelwerte `km3$centers` (die natürlich wie die Daten selbst 8-dimensional sind) zunächst auf die ersten beiden Hauptkomponenten projiziert werden muessen. Dies geschieht wieder mit Hilfe der `predict`-Methode für `"prcomp"`-Objekte. Als Plotsymbol verwenden wir dabei `pch = 3`, vergrößern dies etwas (`cex = 3`) und färben diese bezüglich der Cluster ein (`col = 1:3`).

```
R> gsa.pca <- prcomp(gsa, scale = TRUE)
R> plot(predict(gsa.pca)[, 1:2], type = "n")
R> text(predict(gsa.pca)[, 1:2], rownames(gsa), col = km3$cluster)
R> points(predict(gsa.pca, km3$centers)[, 1:2], col = 1:3, pch = 3,
+      cex = 3)
```

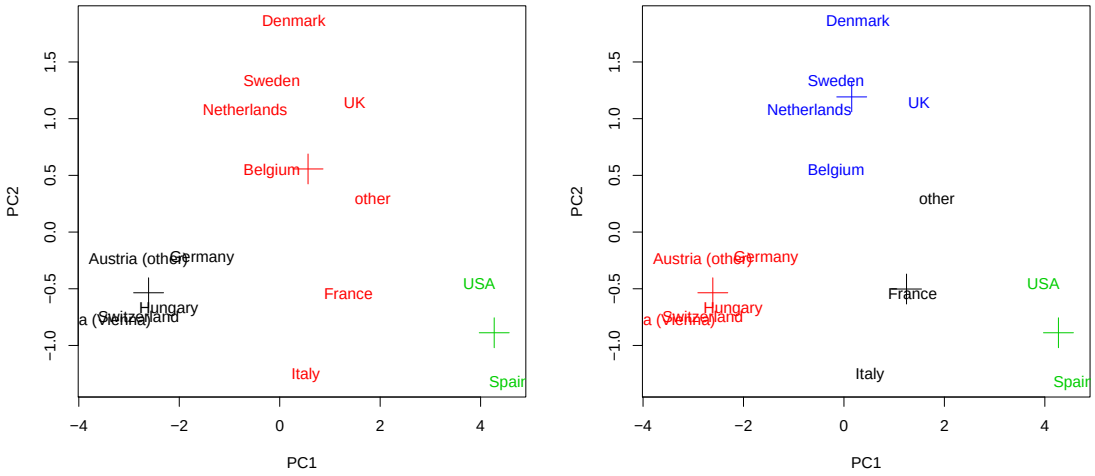


Zur bequemeren Bedienung wurde diese Befehle in der Funktion `plot.kmeans()` zusammengefaßt, die per

```
R> source("plot.kmeans.R")
```

geladen werden kann. Die Funktion nimmt als erstes Argument das angepaßte `"kmeans"`-Objekt und als zweites die Daten, die visualisiert werden sollen. Um die besten 3- und 4-Clusterlösung zu vergleichen, können also einfach folgende Befehle verwendet werden:

```
R> plot(km3, gsa)
R> plot(km4, gsa)
```



Der mittlere Cluster wird also nochmal in zwei unterschiedliche Cluster aufgeteilt. Der erste Cluster in `km3` und `km4` ist identisch, dies sind die bereits oben diskutierten sportinteressierten Touristen. Der dritte Cluster in `km3` ist identisch mit dem vierten Cluster in `km4`, dieser enthält nur Spanien und die USA, die extrem interessiert an Kultur und extrem uninteressiert an Sport sind. Der zweite

Cluster in `km3` wird aufgespaltet in den zweiten und dritten Cluster in `km4`. Der zweite Cluster in `km3` zeichnet sich höchstens durch ein großes Desinteresse an Sport und ein gewisses Interesse an Shopping aus, aber die meisten Werte liegen recht nah am Durchschnittswert 0. Die Aufteilung in zwei Cluster differenziert dies ein bisschen besser: der zweite Cluster zeichnet sich durch ein erhöhtes Interesse an Konzerten und Museen, aber ein nur durchschnittliches Interesse an bspw. Shopping aus. Der dritte Cluster hingegen möchte vor allem für Shopping nach Wien kommen.

```
R> km3$centers
```

```
  SA01.tennis SA02.cycle  SA03.ride   SA05.swim  SA17.shop SA18.concert
1   0.9934126  1.1313181  1.1531649  0.81356723 -1.0913537   -0.8321186
2  -0.2944878 -0.3516905 -0.5124076 -0.04426046  0.3849569    0.1173134
3  -1.3055803 -1.4215332 -0.8332818 -1.85687623  1.1885565    1.6110430
   SA19.sight SA21.museum
1 -0.81223259  -0.5694531
2  0.02556571  -0.1112533
3  1.92831865   1.8686458
```

```
R> km4$centers
```

```
  SA01.tennis SA02.cycle  SA03.ride  SA05.swim  SA17.shop SA18.concert
1 -0.05920148 -0.3697659 -0.4484939 -0.3863731  0.1778420    0.9092146
2  0.99341258  1.1313181  1.1531649  0.8135672 -1.0913537   -0.8321186
3 -1.30558032 -1.4215332 -0.8332818 -1.8568762  1.1885565    1.6110430
4 -0.43565956 -0.3408453 -0.5507558  0.1610071  0.5092259   -0.3578273
  SA19.sight SA21.museum
1  0.4711044   0.7324817
2 -0.8122326  -0.5694531
3  1.9283186   1.8686458
4 -0.2417575  -0.6174942
```