Finding k for k-means clustering

Arjun Poddar

January 01, 2016

k-means clustering is a very popular method to find clusters in a set of data.

In k-means clustering these n data points or vectors are devided or grouped into k groups or "clusters" based on how similar the data points are measured by the d-dimensions. Generally, this "similarity" between the points are calculated by using distance functions, for example the Euclidean distance.

Suppose we have a set of n data points(observations) $(x_1, x_2, \ldots, x_n)$, where each x_i is a point in a d-dimensional space. This means that x_i is the vector $(x_{i1}, x_{i2}, \ldots, x_{id})$, for $i = 1, 2, \ldots, n$. Let there be k clusters, namely $S_1, S_2, \ldots, S_k$. We define the following quantities:

$$\bar{x}_{.j} = \frac{1}{n}\sum_{i=1}^{n} x_{ij}, \text{ the mean of the } j^{th} \text{ component for the entire data}, j = 1, 2, \ldots, d.$$

$$\mu_{lj} = \frac{1}{n_l}\sum_{x_i \in S_l} x_{ij}, \text{ the mean(center) of the } j^{th} \text{ component in the } l^{th} \text{ cluster}, j = 1, 2, \ldots, d,\ l = 1, 2 \ldots, k.$$

The **total sums of the sqaures** (TSS) of the data is given by, $\text{TSS} = \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - \bar{x}_{.j})^2$

Therefore, based on the centers of the k clusters, TSS can be written as

$$\sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - \bar{x}_{.j})^2 = \sum_{l=1}^{k}\sum_{x_i \in S_k}\sum_{j=1}^{d}(x_{ij} - \mu_{lj})^2 + \sum_{l=1}^{k}\sum_{j=1}^{d}(\bar{x}_{.j} - \mu_{lj})^2$$

$\sum_{l=1}^{k}\sum_{x_i \in S_k}\sum_{j=1}^{d}(x_{ij} - \mu_{lj})^2$is called the **Within Cluster Sum of Squares** (WCSS) as it measures the sum of square of the distances of all the points from the respective centers of the clusters they belong to.

$\sum_{l=1}^{k}\sum_{j=1}^{d}(\bar{x}_{.j} - \mu_{lj})^2$is called the **Between Cluster Sum of Squares** (BCSS) as it measures the sum of square of the distances of all the centers of the clusters from the center of the entire data.

Therefore, TSS = BCSS + WCSS

When the value of k is known, here is, in a nutshell, how the k-means clustering algorithm works:

- **Assignment**: Each of the n points of data are assigned to any of the k groups in such a way that the WCSS is minimized.
- **Update**: The centers for all the clusters are updated using the points in allocated to them.

In R, the function kmeans can be used to perform k-means clustering on a data set. The two primary arguements of this function are namely x and centers, where x is the numeric data in appropriate form (points or observations arranged in rows and the different dimensions of the observations in columns) and centers is k(the number of clusters) or a set of initial seed for the distinct k centers.

Finding the right k

If the number of clusters is known, k-means clustering is easy to do. But more often than not, k is unknown because the distribution of the data may be unknown. If k is too small with respect to the size of the data, clustering may not capture the true heterogeneity present in the data. On the other hand, if k is too large a number, using the clustering information may not be useful at all. So, we have to find a balance.

If we change k, WCSS changes (for obvious reasons) but TSS does not (ofcourse). Let us verify this with a simple R code. We will use the numeric columns from the famous Iris data set.

```
# set the data
df <- iris[, 1:4]
#check the data
head(df)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

```
# k-means clustering with k = 7
iris.cluster.7 <- kmeans(df, 7)
# k-means clustering with k = 11
iris.cluster.11 <- kmeans(df, 11)

#check the TSSs from the two clusterings
cbind(iris.cluster.7$totss, iris.cluster.11$totss)
```

```
##          [,1]     [,2]
## [1,] 681.3706 681.3706
```

```
#check the WCSSs from the two clusterings
cbind(iris.cluster.7$tot.withinss, iris.cluster.11$tot.withinss)
```

```
##          [,1]     [,2]
## [1,] 35.99841 27.88145
```

```r
#check the BCSSs from the two clusterings
cbind(iris.cluster.7$betweenss, iris.cluster.11$betweenss)
```

```
##          [,1]     [,2]
## [1,] 645.3722 653.4892
```

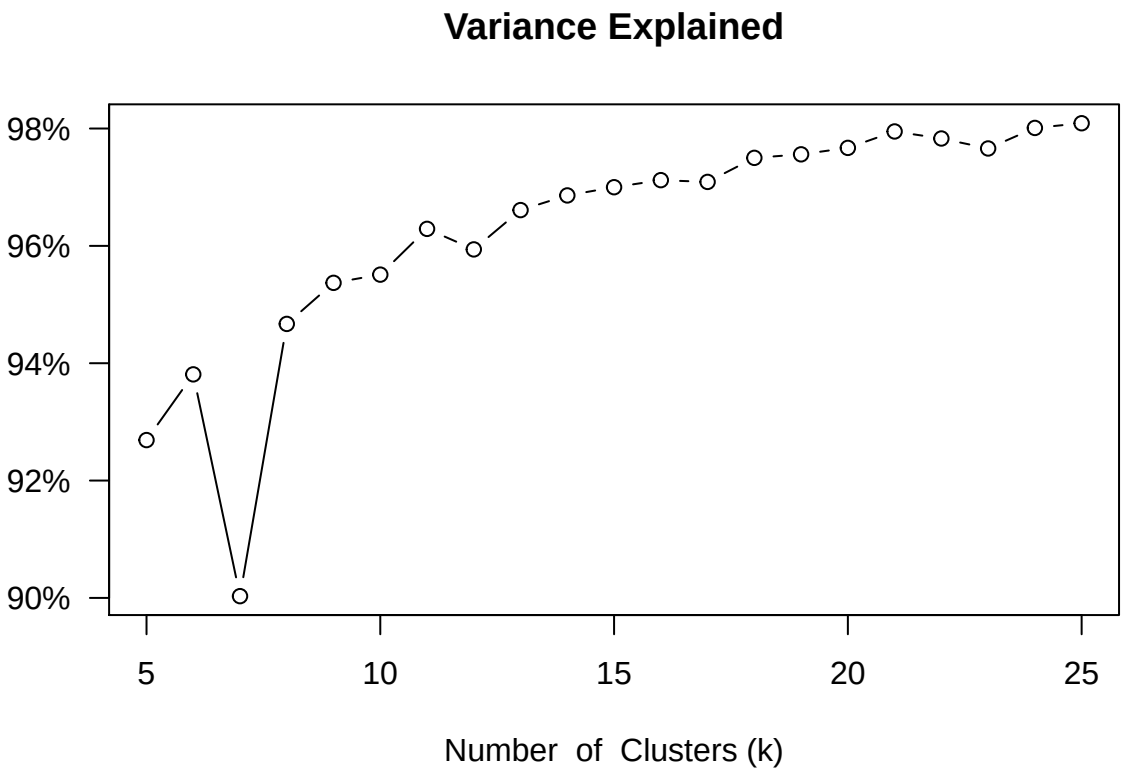
From the last three outputs, it is clear that though the TSS is constant regardless of what k we choose, WCSS and BCSS are not. For each k-means clustering WCSS is calculated by calculating the sums of squares of distance of each point from the center of the cluster it belongs to. BCSS is obtained from subtracting WCSS from TSS.

A smaller WCSS implies a better clustering of the data points. **So to choose a k that gives us better result, we might look for a smaller WCSS or a bigger BCSS. As a criteria to choose the best k, we will run the k-means clustering for different values of k and choose the one which has the highest BCCS** and we will plot the BCSS/TSS. We choose BCSS instead of WCSS because **the ratio BCSS/TSS is equivalent to R^2, coeffieint of determination**. We call this ratio "variance explained" by the clustering.

So we write a function in R named cluster.k which takes three arguements - df(the data frame), k.min (minimum value of k) and k.max(maximum value of k). The default value for k.min is 2 and for k.max is $ceiling(\sqrt{(n/2)})$.

When we run this function on a dataset, a plot of percentage of variance explained and k is produced along with a datafrane named variance.explained and a numeric value called k for which the variance is maximized.

```r
cluster.k(iris[,1:4], 5, 25)
```




```
## The value of k that maximizes the variance explained by clustering is 25
```

```
## $variance.explained
##     k Variance Explained(%)
## 1   5                 92.69
## 2   6                 93.81
## 3   7                 90.03
## 4   8                 94.67
## 5   9                 95.37
## 6  10                 95.51
## 7  11                 96.29
## 8  12                 95.94
## 9  13                 96.61
## 10 14                 96.86
## 11 15                 97.00
## 12 16                 97.12
## 13 17                 97.09
## 14 18                 97.50
## 15 19                 97.56
## 16 20                 97.67
## 17 21                 97.95
## 18 22                 97.83
## 19 23                 97.66
## 20 24                 98.01
## 21 25                 98.09
##
## $k
## [1] 25
```