Lecture 11

Finding Groups of Data – Clustering with k-means

What is clustering?

Clustering is the process of partitioning a set of data objects (or observations) into subsets. Each subset is a **cluster**, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters. The definition of similarity might vary across algorithms.

Clustering has been widely used in many applications. In business intelligence, for example, clustering can be used to organize a large number of customers into groups, where customers within a group share strong similar characteristics. In image recognition, clustering can be used to discover clusters or “subclasses” in handwritten character recognition systems.

Because a cluster is a collection of data objects that are similar to one another within the cluster and dissimilar to objects in other clusters, a cluster of data objects can be treated as an implicit class. In this sense, clustering is sometimes called **automatic classification**. Clustering, however, is different from standard classification in that cluster labels are inferred automatically, i.e., it classifies unlabeled examples, and can discover unknown classes. Because clustering works without advance knowledge of how classes look like and how many classes there are, clustering is used for **knowledge discovery**.

What is clustering?

Because clustering can classify unlabeled examples it performs **unsupervised classification.** For this reason, clustering is a form of **learning by observation**, rather than *learning by examples*.

Clustering can also be used for **outlier detection**, where outliers (values that are “far away” from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

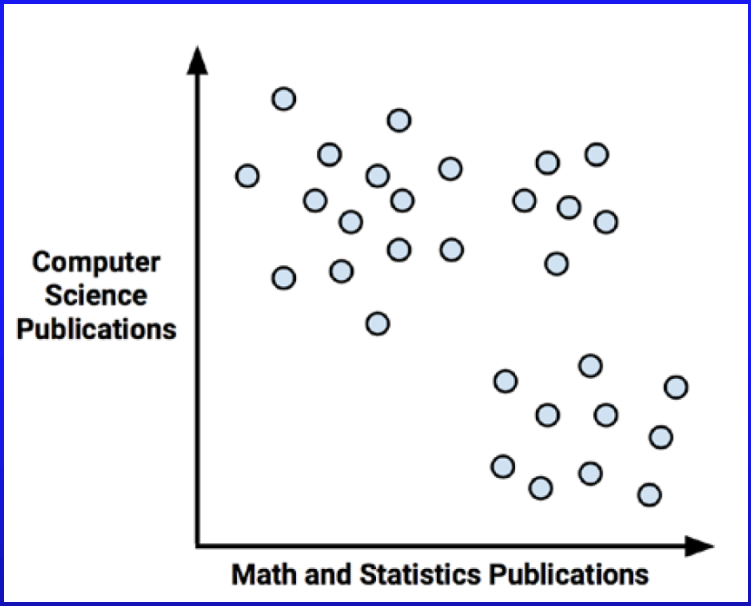
One problem with clustering is that it might be difficult to interpret the meaning of every class. Some classes can be spurious.

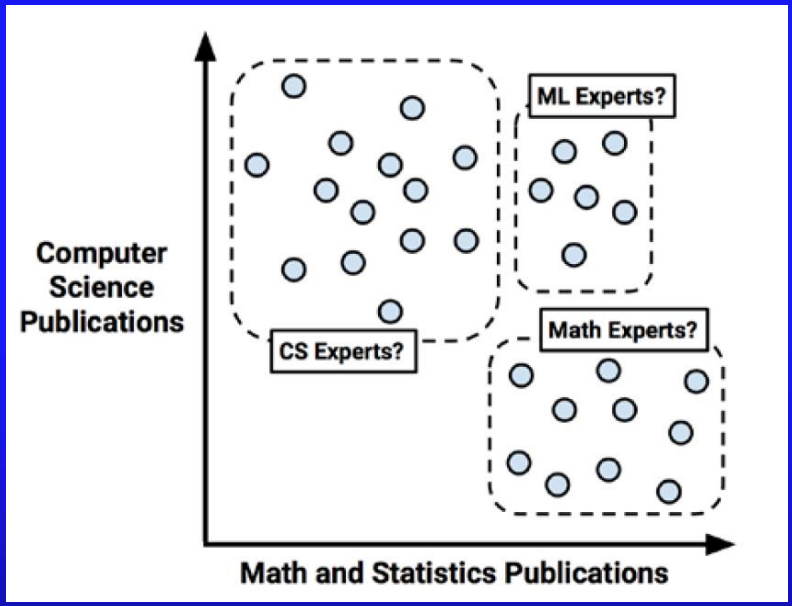
There is a simple example in your textbook of using clustering for classifying academic researchers based on their publication record (the subject of their publications) into three classes: math and statistics, computer science, and ML.

The scatterplot of researchers’ types is shown on the next slide, where each point represents a researcher.

What is clustering?

The data can be visually divided into three clusters shown on the right:

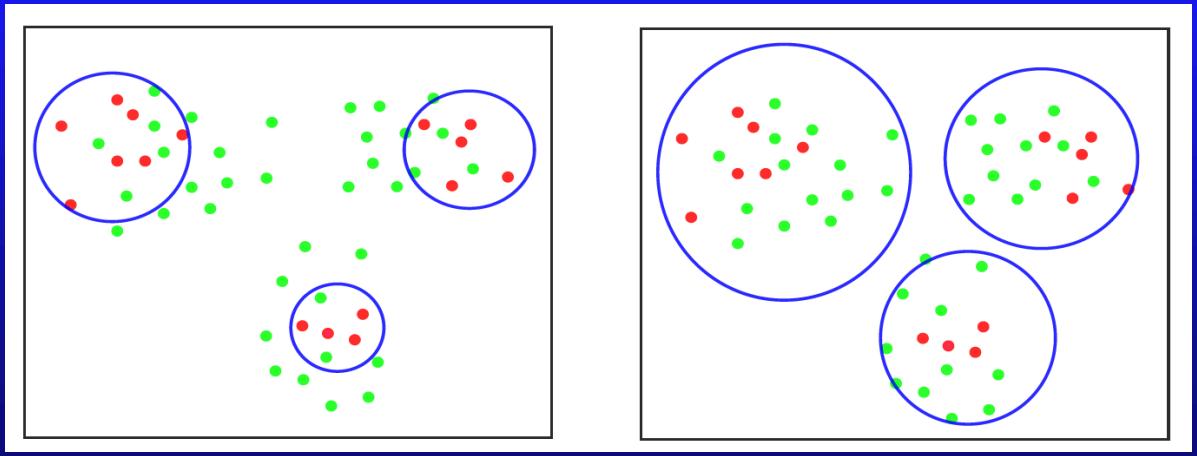




Semi-supervised clustering

There is a branch of clustering called semi-supervised clustering, which uses a large set of unlabeled examples and a small set of labeled data that can be used to guide the clustering process. For example, the labeled examples could be used to bias the search clustering algorithm toward the solutions more consistent with the labeled data, or the labeled data can be used to improve the quality of the result reducing the algorithm natural bias.

In the standard K-means clustering, the initial cluster centers are chosen randomly. In semi-supervised K-means clustering, the labeled data can be used to suggest the location and the number of the clusters. The figure below shows how the clusters are built around the labeled data (red points). Green points represent unlabeled data.



Basic clustering methods

There are many clustering algorithms in the literature. In general, the major fundamental clustering methods can be classified into the following categories:

- Partitioning methods
- Hierarchical methods
- Density-based methods
- Grid-based methods

Partitioning methods construct *k* partitions of the data for a given a set of *n* objects, where each partition represents a cluster and $k \leq n$. That is, they divide the data into *k* groups such that each group must contain at least one object. In other words, partitioning methods conduct one-level partitioning on data sets. The basic partitioning methods typically adopt *exclusive cluster separation*. That is, each object must belong to exactly one group. This requirement may be relaxed, for example, in fuzzy partitioning techniques.

Basic clustering methods

Most partitioning methods are distance-based. Given *k*, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an **iterative relocation technique** that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects in different clusters are "far apart" or very different.

Achieving global optimality in partitioning-based clustering is often computationally prohibitive, potentially requiring an exhaustive enumeration of all the possible partitions. Instead, most applications adopt popular heuristic methods, such as greedy approaches like the *k-means* and the *k-medoids* algorithms, which progressively improve the clustering quality and approach a local optimum. These heuristic clustering methods work well for finding spherical-shaped clusters in small- to medium-size databases.

Basic clustering methods

A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the *bottom-up* approach, starts with each object forming a separate group. It successively merges the objects or groups close to one another, until all the groups are merged into one (the topmost level of the hierarchy), or a termination condition holds. The *divisive approach*, also called the *top-down* approach, starts with all the objects in the same cluster. In each successive iteration, a cluster is split into smaller clusters, until eventually each object is in one cluster, or a termination condition holds.

Hierarchical clustering methods can be distance-based or density- and continuity based.

A typical problem for hierarchical methods is that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices.

Basic clustering methods

Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty in discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of ***density***. Their general idea is to continue growing a given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold. For example, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise or outliers and discover clusters of arbitrary shape.

Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All the clustering operations are performed on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.

The k-means clustering algorithm

A brief summary of the algorithm: First, you specify in advance how many clusters are being sought: This is the parameter *k*. Then *k* points are chosen at random as cluster centers. All instances are assigned to their closest cluster center according to some distance metric, e.g., Euclidean. Next the *centroid*, or mean, of the instances in each cluster is calculated—this is the "means" part. These centroids are taken to be new center values for their respective clusters. Finally, the whole process is repeated with the new cluster centers. Iteration continues until the same points are assigned to each cluster in consecutive rounds, at which stage the cluster centers have stabilized and will remain the same forever. In other words k–means is a partitioning clustering algorithm that performs iterative distance-based clustering .

Suppose a data set, *D*, contains *n* objects in Euclidean space. Partitioning methods distribute the objects in *D* into *k* clusters, $C_1, \ldots, C_k$, that is, $C_i \subset D$ and $C_i \cap C_j = \emptyset$. An objective function is used to assess the partitioning quality so that objects within a cluster are similar to one another but dissimilar to objects in other clusters. This is, the objective function aims for high intracluster similarity and low intercluster similarity.

The k-means clustering algorithm

A centroid-based partitioning technique uses the *centroid* of a cluster, C_i , to represent that cluster. Conceptually, the centroid of a cluster is its center point. The centroid can be defined in various ways such as by the mean or medoid of the objects (or points) assigned to the cluster. The difference between an object $\boldsymbol{p} \in C_i$ and $\boldsymbol{c_i}$, the representative of the cluster, is measured by $dist(\boldsymbol{p}, \boldsymbol{c_i})$, where $dist(\boldsymbol{x},\boldsymbol{y})$ is the Euclidean distance between two points $\boldsymbol{x}$ and $\boldsymbol{y}$. The quality of cluster C_i can be measured by the **withincluster variation**, which is the sum of *squared error* between all objects in C_i and the centroid $\boldsymbol{c_i}$, defined as:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(\boldsymbol{p}, \boldsymbol{c_i})^2,$$

where E is the sum of the squared error for all objects in the data set; $\boldsymbol{p}$ is the point in space representing a given object; and $\boldsymbol{c_i}$ is the centroid of cluster C_i (both $\boldsymbol{p}$ and $\boldsymbol{c_i}$ are multidimensional).

The k-means clustering algorithm

Optimizing the within-cluster variation is computationally challenging. In the worst case, we would have to enumerate a number of possible partitionings that are exponential to the number of clusters, and check the within-cluster variation values. It has been shown that the problem is NP-hard in general Euclidean space even for two clusters. Moreover, the problem is NP-hard for a general number of clusters *k* even in the 2-D Euclidean space. The *k*-means algorithm is summarized as follows:

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

The k-means clustering algorithm

The *k*-means method is not guaranteed to converge to the global optimum and often terminates at a local optimum. The results may depend on the initial random selection of cluster centers. The time complexity of the *k*-means algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations.

The *k*-means method can be applied only when the mean of a set of objects is defined. This may not be the case in some applications such as when data with nominal attributes are involved. The ***k*-modes method** is a variant of *k*-means, which extends the *k*-means paradigm to cluster nominal data by replacing the means of clusters with modes. It uses new dissimilarity measures to deal with nominal objects and a frequency-based method to update modes of clusters. The *k*-means and the *k*-modes methods can be integrated to cluster data with mixed numeric and nominal values.

Problems with k-means

In addition to the problem of finding the initial mean values (i.e., the final clusters are sensitive to the initial cluster centers), another problem is the necessity to specify *k*, the number of clusters, in advance. There have been studies on how to overcome this difficulty, however, such as by providing an approximate range of *k* values, and then using an analytical technique to determine the best *k* by comparing the clustering results obtained for the different *k* values. The *k*-means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size. It works best for spherical sizes when the Euclidean distance is used. Moreover, k-means is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value. The *k*-means algorithm is sensitive to outliers because such objects are far away from the majority of the data, and thus, when assigned to a cluster, they can dramatically distort the mean value of the cluster. K-means can be modified in order to diminish such sensitivity to outliers*:* Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is assigned to the cluster of which the representative object is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object ***p*** and its corresponding representative object.

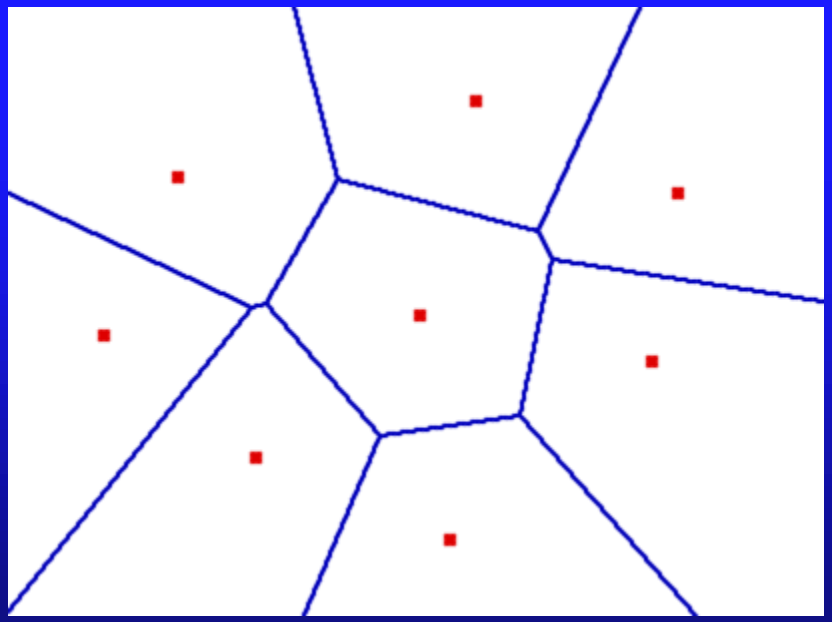
An example of suboptimal solution

It is easy to imagine situations in which *k*-means fails to find a good clustering. Consider four instances arranged at the vertices of a rectangle in two-dimensional space. There are two natural clusters, formed by grouping together the two vertices at either end of a short side. But suppose the two initial cluster centers happen to fall at the midpoints of the *long* sides. This forms a stable configuration. The two clusters each contain the two instances at either end of a long side—no matter how great the difference between the long and the short sides.

k-means clustering can be dramatically improved by careful choice of the initial cluster centers, often called *seeds*. Instead of beginning with an arbitrary set of seeds, here is a better procedure. Choose the initial seed at random from the entire space, with a uniform probability distribution. Then choose the second seed with a probability that is proportional to the square of the distance from the first. Proceed, at each stage choosing the next seed with a probability proportional to the square of the distance from the closest seed that has already been chosen. This procedure, called *k-means*++, improves both speed and accuracy over the original algorithm with random seeds.

Voronoi diagram

If we look at a random (x,y) coordinate in the cluster graph, we will find that one cluster is closest. Thus we can color each (x,y) point (each pixel) with the color of its assigned cluster. The result is a diagram that divides the whole space into cells. The borders between cells are exactly half-way between the two closest cluster means. This kind of diagram is called a Voronoi diagram (named after Georgy Voronoi).



Determining the number of clusters

The k-means algorithm is sensitive not only to the randomly-chosen cluster centers, but also to the number of clusters. Determining the “right” number of clusters in a data set is important because the appropriate number of clusters controls the proper granularity of cluster analysis. It can be regarded as finding a good balance between *compressibility* and *accuracy* in cluster analysis. Consider two extreme cases. What if you were to treat the entire data set as a cluster? This would maximize the compression of the data, but such a cluster analysis has no value. On the other hand, treating each object in a data set as a cluster gives the finest clustering resolution (i.e., most accurate due to the zero distance between an object and the corresponding cluster center). However, having one object per cluster does not enable any data summarization.

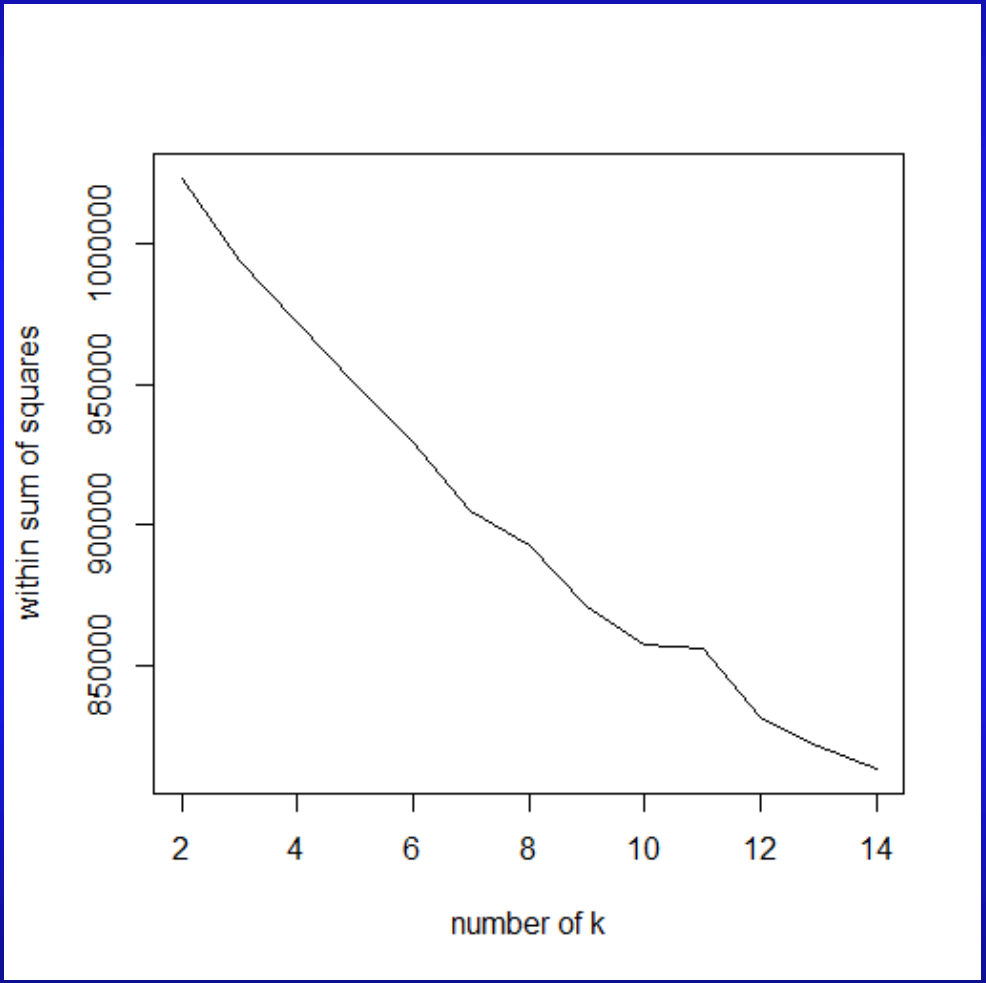
A simple method is to set the number of clusters to about $\sqrt{\frac{n}{2}}$ for a data set of *n* points. In expectation, each cluster has $\sqrt{2n}$ points. However, this rule is likely to result in an unwieldy number of clusters for large datasets.

Determining the number of clusters

The **elbow method** is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster. This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other. However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction. Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Technically, given a number, $k > 0$, we can form k clusters on the data set in question using k-means, and calculate the sum of within-cluster variances. We can then plot the curve of *var* with respect to k. The first (or most significant) turning point of the curve suggests the “right” number.

Determining the number of clusters



There is an elbow for k=10

Clustering quality

There are many methods for evaluating clustering quality. One of them is the **silhouette coefficient**, which belongs to the class of intrinsic methods for evaluating clustering quality. In general, intrinsic methods evaluate a clustering by examining how well the clusters are separated and how compact the clusters are.

For a data set, D, of n objects, suppose D is partitioned into k clusters, $C_1, \ldots, C_k$. For each object $\boldsymbol{o} \in D$, we calculate $a(\boldsymbol{o})$ as the average distance between $\boldsymbol{o}$ and all other objects in the cluster to which $\boldsymbol{o}$ belongs. Similarly, $b(\boldsymbol{o})$ is the minimum average distance from $\boldsymbol{o}$ to all clusters to which $\boldsymbol{o}$ does not belong. Formally, suppose $\boldsymbol{o} \in C_i$ $(1 \le l \le k)$. Then:

$$a(\boldsymbol{o}) = \frac{\sum_{\boldsymbol{o}' \in C_i, \boldsymbol{o} \neq \boldsymbol{o}'} dist(\boldsymbol{o}, \boldsymbol{o}')}{|C_i| - 1}$$

$$b(\boldsymbol{o}) = \min_{C_j : 1 \le j \le k, j \neq i} \left\{ \frac{\sum_{\boldsymbol{o}' \in C_j} dist(\boldsymbol{o}, \boldsymbol{o}')}{|C_j|} \right\}.$$

Clustering quality

The **silhouette coefficient** of ***o*** is defined as:

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}.$$

The value of the silhouette coefficient is between -1 and 1. The value of ***a***(***o***) reflects the compactness of the cluster to which ***o*** belongs. The smaller the value, the more compact the cluster. The value of *b*(***o***) captures the degree to which ***o*** is separated from other clusters. The larger *b*(***o***) is, the more separated ***o*** is from other clusters. Therefore, when the silhouette coefficient value of ***o*** approaches 1, the cluster containing ***o*** is compact and ***o*** is far away from other clusters, which is the preferable case. However, when the silhouette coefficient value is negative (i.e., *b*(***o***) < a(***o***)), this means that, in expectation, ***o*** is closer to the objects in another cluster than to the objects in the same cluster as ***o***. In many cases, this is a bad situation and should be avoided. To measure a cluster's fitness within a clustering, we can compute the average silhouette coefficient value of all objects in the cluster. To measure the quality of a clustering, we can use the average silhouette coefficient value of all objects in the data set.

Example – finding teen market segments

We will use the k-means algorithm to identify segments of teenagers who share similar tastes, such as sports, religion, or music. Such segments are often used for online targeting advertisements and profiling. The data is collected from the teenagers' social networking pages. The most interesting aspect of the data is that it has been collected by the author of your textbook, Brett Lantz, while conducting sociological research on the teenage identities at the University of Notre Dam. You can download the file *snsdata.csv* from Blackboard and save it into your working directory.

The data was sampled evenly across four high school graduation years (2006 through 2009) representing the senior, junior, sophomore, and freshman classes at the time of data collection. Using an automated web crawler, the full text of the SNS profiles were downloaded, and each teen's gender, age, and number of SNS friends was recorded.

Example – finding teen market segments

A text mining tool was used to divide the remaining SNS page content into words. From the top 500 words appearing across all the pages, 36 words were chosen to represent five categories of interests: namely extracurricular activities, fashion, religion, romance, and antisocial behavior. The 36 words include terms such as *football*, *sexy*, *kissed*, *bible*, *shopping*, *death*, and *drugs*. The final dataset indicates, for each person, how many times each word appeared in the person's SNS profile.

```
> teens <- read.csv("snsdata.csv")
> str(teens)
'data.frame':   30000 obs. of  40 variables:
 $ gradyear    : int  2006 2006 2006 2006 2006 2006 2006 2006 2006 2006 ...
 $ gender      : Factor w/ 2 levels "F","M": 2 1 2 1 NA 1 1 2 1 1 ...
 $ age         : num  19 18.8 18.3 18.9 19 ...
 $ friends     : int  7 0 69 0 10 142 72 17 52 39 ...
 $ basketball  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ football    : int  0 1 1 0 0 0 0 0 0 0 ...
……..
```

The gender variable has missing values for some individuals.

Exploring and preparing the data

We will use the *table()* function to count the number of NA values:

```
> table(teens$gender, useNA = "ifany")
    F     M  <NA>
22054  5222  2724
```

To see NAs among the table output, you can indicate "ifany" or "always" in the *useNA* argument. The first shows NAs in the output only if there is some missing data. The second will include NAs in the output regardless. The command shows that 2,724 records (9 percent) have missing gender data.

The age variable has also missing values:

```
> summary(teens$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  3.086  16.310  17.290  17.990  18.260 106.900    5086
```

The summary shows that 5086 values of age are missing. Obviously, the minimum and the maximum values of 3.086 and 106.900 are not accurate and the data needs to be cleaned.

Exploring and preparing the data

We will clean the age variable by converting all values falling outside the interval [13,20) into NAs.

```
>  teens$age <- ifelse(teens$age >= 13 & teens$age
+ < 20,teens$age, NA)
```

The result seems more reasonable:

```
> summary(teens$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  13.03   16.30   17.26   17.25   18.22   20.00    5523
```

Dummy coding missing values

We need to clean the missing gender values. We cannot just remove all rows with missing values because of their high number: missing values of gender and age account for *9% + 17% = 26%* of our data. Moreover, there are cases where missing values can correlate with other features or form a pattern of their own.

Instead, we will define a dummy categorical variables for missing gender values, essentially treating missing values as a separate category. The variable *teens$female* will be defined as 1 if the teen's gender is "F" and as 0 if the teen's gender is "M" or NA. The variable *teens$no_gender* is defined as 1 if the teen's gender is NA, and as 0 if the teen's age is either "M" or "F". This way, we get the following combinations:

	teens$female	teens$no_gender
"F"	1	0
NA	0	1
"M"	0	0

Dummy coding missing values

```
> teens$female <- ifelse(teens$gender == "F" &
+ !is.na(teens$gender), 1, 0)
> teens$no_gender <- ifelse(is.na(teens$gender), 1, 0)
```

We can check the results:

```
> table(teens$female)

    0     1
 7946 22054
> table(teens$gender, useNA = "ifany")

    F     M  <NA>
22054  5222  2724
> table(teens$female, useNA = "ifany")

    0     1
 7946 22054
> table(teens$no_gender, useNA = "ifany")

    0     1
27276  2724
```

Imputing the missing values

How to handle the 5,523 missing age values? Using categorical variables for them is not a good solution. Instead, we will perform an **imputation**, i.e., filling in the missing values with a guess as to their true value. Using the average or mean value as a guess will get us only part of the way there; using the average age for each graduation year is better. The average age per graduation year can be displayed using the function *aggregate(),* which can split the data into subsets, and compute summary statistics for each subset.

```
> aggregate(data = teens, age ~ gradyear, mean, na.rm = TRUE)
  gradyear      age
1     2006 19.13724
2     2007 18.39146
3     2008 17.52387
4     2009 16.87602
```

The parameter na.rm = TRUE indicates that all missing values have been excluded from computing the mean. The *aggregate()* function returns a data frame of 4 rows and two variables, which is not very convenient for filling the missing age values. As an alternative, we can use the *ave()* function, which returns a vector with the group means repeated so that the result is equal in length to the original vector of 30000 teens.

Imputing the missing values

```
> ave_age <- ave(teens$age, teens$gradyear, FUN =
+ function(x) mean(x, na.rm = TRUE))
> str(ave_age)
 num [1:30000] 19.1 19.1 19.1 19.1 19.1 ...
```

The actual imputation into the missing values can be done using the *ifelse()* function:

```
> teens$age <- ifelse(is.na(teens$age), ave_age, teens$age)
```

The *summary()* function shows that the missing values have now been eliminated:

```
> summary(teens$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.086  16.500  17.440  17.980  18.390 106.900
```

Training a model on the data

We will use the *kmeans()* function from the *stats* package, which is included in the default R installation.

Clustering syntax

using the `kmeans()` function in the `stats` package

Finding clusters:

```
myclusters <- kmeans(mydata, k)
```

- `mydata` is a matrix or data frame with the examples to be clustered
- `k` specifies the desired number of clusters

The function will return a cluster object that stores information about the clusters.

Examining clusters:

- `myclusters$cluster` is a vector of cluster assignments from the `kmeans()` function
- `myclusters$centers` is a matrix indicating the mean values for each feature and cluster combination
- `myclusters$size` lists the number of examples assigned to each cluster

Example:

```
teen_clusters <- kmeans(teens, 5)
teens$cluster_id <- teen_clusters$cluster
```

Training a model on the data

Since we want to identify teen clusters using the 36 variables that represent the number of times various interests appeared on the teen social profiles, we will exclude the first four variables from the data frame:

```
> interests <- teens[5:40]
```

In other words, interests does not include the variables gradyear, gender, age, and friends.

Before, we can run the k-means algorithm, we need to normalize our data in order to make sure that all variables have the same range. Otherwise differences in variables' ranges can skew the results. We performed normalization earlier in this course when we applied the kNN algorithm.

```
> interests_z <- as.data.frame(lapply(interests, scale))
```

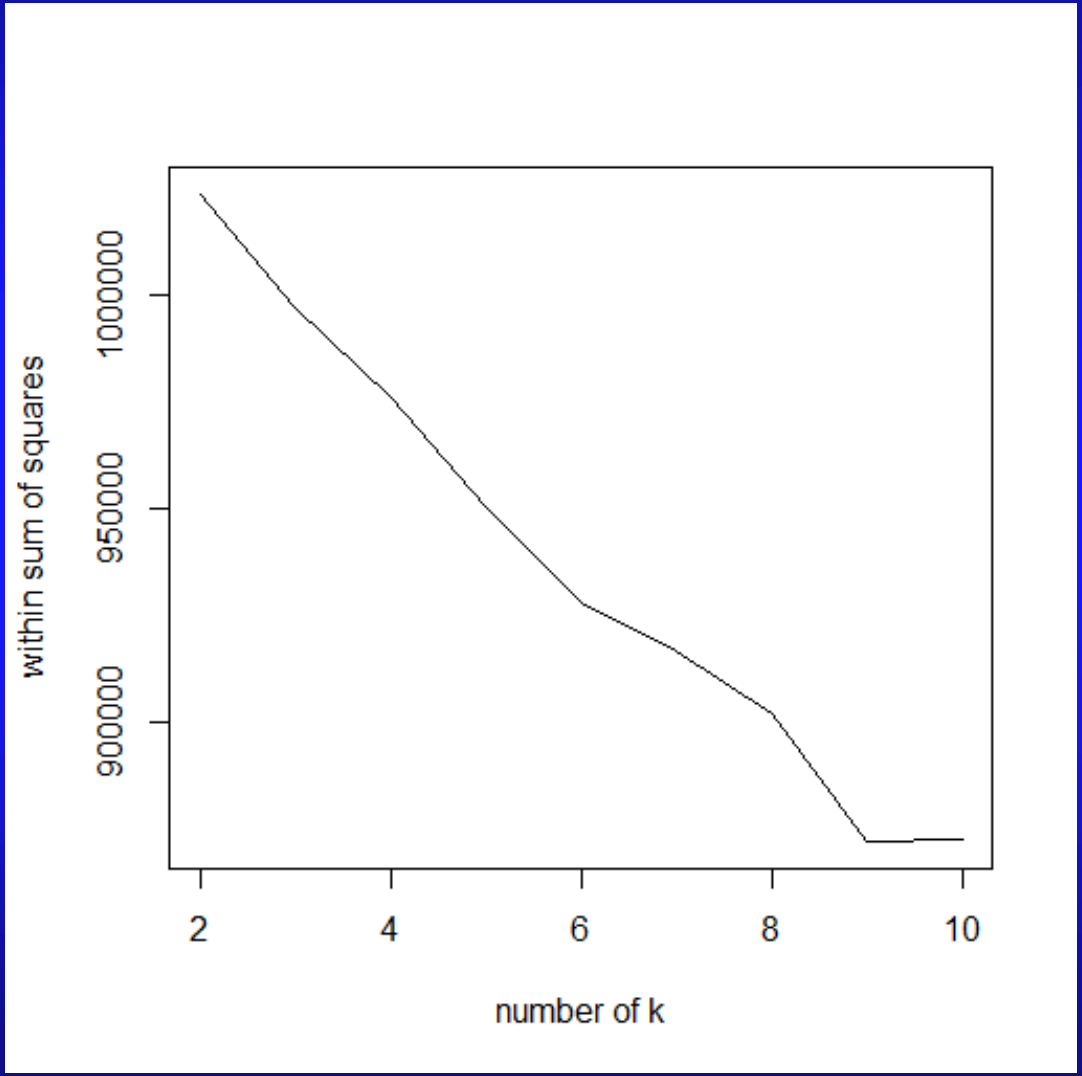
The process of z-score standardization rescales features so that they have a mean of zero and a standard deviation of one.

Choosing the number of clusters

We will use the elbow method to find the number of the clusters, calculating the within sum of squares (withinss) of different numbers of clusters:

```
> nk = 2:14
> set.seed(1)
> WSS = sapply(nk, function(k) {kmeans(interests_z, centers=k)$tot.withinss})
> WSS
 [1] 1023723.4  994080.2  972110.5  950121.3  929519.3  904708.9  892982.1
 [8]  870682.5  856922.9  856103.6  831277.8  820965.0  812845.1
> plot(nk, WSS, type="l", xlab= "number of k", ylab="within sum of squares")
```

Choosing the number of clusters



There is an elbow at k = 6 and anther elbow at k = 9

Evaluating model performance

Your textbook uses k = 5:

```
> set.seed(2345)
> teen_clusters <- kmeans(interests_z, 5)
```

kmeans returns an object of class "kmeans" which has the following components:

- cluster: A vector of integers (from 1:k) indicating the cluster to which each - point is allocated.
- centers: A matrix of cluster centres.
- totss: The total sum of squares.
- withinss: Vector of within-cluster sum of squares, one component per cluster.
- tot.withinss: Total within-cluster sum of squares, i.e. sum(withinss).
- betweenss: The between-cluster sum of squares, i.e. totss-tot.withinss.
- size: The number of points in each cluster.
- iter: The number of (outer) iterations.

Evaluating model performance

The evaluation of the model performance depends on the model goal. In our case, this is identifying clusters of teenagers with similar interests for marketing purposes. First, we would like to examine cluster sizes. If the clusters are too large or too small, they are not likely to be very useful:

```
> teen_clusters$size
[1]   871   600  5981  1034 21514
```

The cluster size of 21,514 (72 percent) could be problematic. Sometimes, k-means may find extremely small clusters, as small as a single point. This can happen if one of the initial cluster centers happens to fall on an outlier far from the rest of the data. To exclude such cases, it may be worth rerunning the k-means algorithm with a different random seed to see whether the small cluster is robust to different starting points.

We can examine the coordinates of the cluster centroids using the teen_clusters$centers component:

```
> teen_clusters$centers
```

The results are shown on the next slide.

Clusters' centers

	basketball	football	soccer	softball	volleyball	swimming
1	0.16001227	0.2364174	0.10385512	0.07232021	0.18897158	0.23970234
2	-0.09195886	0.0652625	-0.09932124	-0.01739428	-0.06219308	0.03339844
3	0.52755083	0.4873480	0.29778605	0.37178877	0.37986175	0.29628671
4	0.34081039	0.3593965	0.12722250	0.16384661	0.11032200	0.26943332
5	-0.16695523	-0.1641499	-0.09033520	-0.11367669	-0.11682181	-0.10595448

	cheerleading	baseball	tennis	sports	cute	sex
1	0.3931445	0.02993479	0.13532387	0.10257837	0.37884271	0.020042068
2	-0.1101103	-0.11487510	0.04062204	-0.09899231	-0.03265037	-0.042486141
3	0.3303485	0.35231971	0.14057808	0.32967130	0.54442929	0.002913623
4	0.1856664	0.27527088	0.10980958	0.79711920	0.47866008	2.028471066
5	-0.1136077	-0.10918483	-0.05097057	-0.13135334	-0.18878627	-0.097928345

	sexy	hot	kissed	dance	band	marching
1	0.11740551	0.41389104	0.06787768	0.22780899	-0.10257102	-0.10942590
2	-0.04329091	-0.03812345	-0.04554933	0.04573186	4.06726666	5.25757242
3	0.24040196	0.38551819	-0.03356121	0.45662534	-0.02120728	-0.10880541
4	0.51266080	0.31708549	2.97973077	0.45535061	0.38053621	-0.02014608
5	-0.09501817	-0.13810894	-0.13535855	-0.15932739	-0.12167214	-0.11098063

	music	rock	god	church	jesus	bible
1	0.1378306	0.05905951	0.03651755	-0.00709374	0.01458533	-0.03692278
2	0.4981238	0.15963917	0.09283620	0.06414651	0.04801941	0.05863810
3	0.2844999	0.21436936	0.35014919	0.53739806	0.27843424	0.22990963
4	1.1367885	1.21013948	0.41679142	0.16627797	0.12988313	0.08478769
5	-0.1532006	-0.12460034	-0.12144246	-0.15889274	-0.08557822	-0.06813159

	hair	dress	blonde	mall	shopping	clothes
1	0.43807926	0.14905267	0.06137340	0.60368108	0.79806891	0.5651537331
2	-0.04484083	0.07201611	-0.01146396	-0.08724304	-0.03865318	-0.0003526292
3	0.23612853	0.39407628	0.03471458	0.48318495	0.66327838	0.3759725120
4	2.55623737	0.53852195	0.36134138	0.62256686	0.27101815	1.2306917174
5	-0.20498730	-0.14348036	-0.02918252	-0.18625656	-0.22865236	-0.1865419798

	hollister	abercrombie	die	death	drunk	drugs
1	4.1521844	3.96493810	0.043475966	0.09857501	0.035614771	0.03443294
2	-0.1678300	-0.14129577	0.009447317	0.05135888	-0.086773220	-0.06878491
3	-0.0553846	-0.07417839	0.037989066	0.11972190	-0.009688746	-0.05973769
4	0.1610784	0.26324494	1.712181870	0.93631312	1.897388200	2.73326605
5	-0.1557662	-0.14861104	-0.094875180	-0.08370729	-0.087520105	-0.11423381

Evaluating model performance

	basketball	football	soccer	softball	volleyball	swimming
1	0.16001227	0.2364174	0.10385512	0.07232021	0.18897158	0.23970234
2	-0.09195886	0.0652625	-0.09932124	-0.01739428	-0.06219308	0.03339844
3	0.52755083	0.4873480	0.29778605	0.37178877	0.37986175	0.29628671
4	0.34081039	0.3593965	0.12722250	0.16384661	0.11032200	0.26943332
5	-0.16695523	-0.1641499	-0.09033520	-0.11367669	-0.11682181	-0.10595448

	cheerleading	baseball	tennis	sports	cute	sex
1	0.3931445	0.02993479	0.13532387	0.10257837	0.37884271	0.020042068
2	-0.1101103	-0.11487510	0.04062204	-0.09899231	-0.03265037	-0.042486141
3	0.3303485	0.35231971	0.14057808	0.32967130	0.54442929	0.002913623
4	0.1856664	0.27527088	0.10980958	0.79711920	0.47866008	2.028471066
5	-0.1136077	-0.10918483	-0.05097057	-0.13135334	-0.18878627	-0.097928345

	sexy	hot	kissed	dance	band	marching	music
1	0.11740551	0.41389104	0.06787768	0.22780899	-0.10257102	-0.10942590	0.1378306
2	-0.04329091	-0.03812345	-0.04554933	0.04573186	4.06726666	5.25757242	0.4981238
3	0.24040196	0.38551819	-0.03356121	0.45662534	-0.02120728	-0.10880541	0.2844999
4	0.51266080	0.31708549	2.97973077	0.45535061	0.38053621	-0.02014608	1.1367885
5	-0.09501817	-0.13810894	-0.13535855	-0.15932739	-0.12167214	-0.11098063	-0.1532006

Evaluating model performance

The rows of the output (labeled 1 to 5) refer to the five clusters, while the numbers across each row indicate the cluster's average value for the interest listed at the top of the column. As the values are z-score standardized, positive values are above the overall mean level for all the teens and negative values are below the overall mean.

The author divides the teens in five groups, called brains, athletes, basket cases, princesses, and criminals after the movie “The *Breakfast Club”*, a comedy released in 1985.

The results show that Cluster 3 is above the mean interest level on all the sports. This suggests that this may be the group of athletes per *The Breakfast Club* movie. Cluster 1 includes the most mentions of "cheerleading," the word "hot," and is above the average level of football interest. The cluster is called Princesses. The members of Cluster 5 have lower-than-average levels of interest in every measured activity. This is also the single largest group in terms of the number of members. One potential explanation is that these users created a profile on the website but never posted any interests. By continuing to examine the clusters in this way, it is possible to construct a table listing the dominant interests of each of the groups.

Table of teens clusters according to their interests

Cluster 1 (N = 3,376)	Cluster 2 (N = 601)	Cluster 3 (N = 1,036)	Cluster 4 (N = 3,279)	Cluster 5 (N = 21,708)
swimming cheerleading cute sexy hot dance dress hair mall hollister abercrombie shopping clothes	band marching music rock	sports sex sexy hot kissed dance music band die death drunk drugs	basketball football soccer softball volleyball baseball sports god church Jesus bible	???
Princesses	Brains	Criminals	Athletes	Basket Cases

Analysis of findings

The teen_clusters object created by the *kmeans()* function includes a component named cluster that contains the cluster assignments for all 30,000 individuals in the sample. We can add this as a column on the teens data frame:

```
> teens$cluster <- teen_clusters$cluster
```

Using the *aggregate()* function, we can also look at the demographic characteristics of the clusters:

```
> aggregate(data = teens, age ~ cluster, mean)
  cluster      age
1       1 16.86497
2       2 17.39037
3       3 17.07656
4       4 17.11957
5       5 17.29849
```

This shows that age does not vary much by cluster.

Analysis of findings

```
> aggregate(data = teens, female ~ cluster, mean)
  cluster    female
1       1 0.8381171
2       2 0.7250000
3       3 0.8378198
4       4 0.8027079
5       5 0.6994515
```

Cluster 1, the Princesses, is nearly 84 percent female, while Cluster 2 and Cluster 5 are only about 70 percent female. These disparities imply that there are differences in the interests that teen boys and girls discuss on their social networking pages.

Analysis of findings

```
>  aggregate(data = teens, friends ~ cluster, mean)
  cluster  friends
1       1 41.43054
2       2 32.57333
3       3 37.16185
4       4 30.50290
5       5 27.70052
```

On an average, Princesses have the most friends (41.4), followed by Athletes (37.2) and Brains (32.6). On the low end are Criminals (30.5) and Basket Cases (27.7).

The association among group membership, gender, and number of friends suggests that the clusters can be useful predictors of behavior.