Clustering

Hoogendoorn, Mark; Funk, Burkhardt

published in
Machine Learning for the Quantified Self
2018

DOI (link to publisher)
10.1007/978-3-319-66308-1_5

document version
Publisher's PDF, also known as Version of record

Chapter 5
Clustering

This chapter is devoted to techniques that can provide us with insights in the data, namely whether we can find some structure in terms of clusters in the data. For instance, we might want to identify clusters of locations often visited by Bruce to see what impact a specific location has on his mood. You could also be interested in finding clusters of points that identify different levels of activity for Arnold. Another option is finding clusters of like-minded people, that way we could offer them feedback and support, which seems to work well for their fellow clustermen. We will treat such clustering algorithms in this chapter. The membership of data points or people in a certain cluster might in turn become an attribute for our predictive models later on. We will start by discussing the learning setup.

5.1 Learning Setup

We should consider our specific setting of the quantified self before we can start to apply clustering algorithms. You can imagine that there are many people that generate datasets in the quantified self; there might be a lot of "wannabe Arnold's" and a lot of "please do not let me become Bruce" people out there. We will refer to the datasets of n specific people by means of the notation $qs_1, \ldots, qs_n$ resulting in their datasets $\mathbf{X_{qs_1}}, \ldots, \mathbf{X_{qs_n}}$. Furthermore, let x_{i,qs_j} denote the ith data point of the jth person. When it comes to clustering, we have two levels on which we can cluster: *individual data points* and the level of a *person*. Let us consider an example to make the distinction clear. Assume that Arnold is generating data with his accelerometer. We might be interested to find clusters of data points with respect to the accelerometer data that represent different activities (e.g. a cluster for walking, jogging, cycling, etc.). This would be clustering over individual data points. Of course, you might take the union of data sets of multiple people as a basis for this type of cluster if desired. On the other hand, we might be interested in defining types of people we collect data

M. Hoogendoorn and B. Funk, *Machine Learning for the Quantified Self*,
Cognitive Systems Monographs 35, https://doi.org/10.1007/978-3-319-66308-1_5

from, e.g. a cluster for people who share similar characteristics as Arnold or Bruce. To make this more formal, the points in the clustering space we consider for the two scenarios are:

- *individual data points*:

$$\mathbf{X} = \begin{bmatrix} x_{1,qs_1} \\ \vdots \\ x_{N,qs_1} \\ \vdots \\ x_{1,qs_n} \\ \vdots \\ x_{N,qs_n} \end{bmatrix}$$

- *person*:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X_{qs_1}} \\ \vdots \\ \mathbf{X_{qs_n}} \end{bmatrix}$$

Obviously, this choice has an impact on how to define the distance between the points in the clustering space. We will see this in the next section.

5.2 Distance Metrics

Clustering algorithms in general work with a notion of distance between points. We will look at the distances between points for our different setups of our clustering problem.

5.2.1 Individual Data Points Distance Metrics

For individual data points there are a lot of commonly used distance metrics. It depends on the nature of the data which one would be appropriate to use. In case we only have numerical data we can use two of the most well known metrics, the *Euclidean* and the *Manhattan* distance. The two distances between data points x_i and x_j are defined as follows:

$$euclidean\_distance(x_i, x_j) = \sqrt{\sum_{k=1}^{p}(x_i^k - x_j^k)^2} \tag{5.1}$$

$$manhattan\_distance(x_i, x_j) = \sum_{k=1}^{p}|x_i^k - x_j^k| \tag{5.2}$$

The Euclidean distance corresponds to what we typically just call the distance between two points. The Manhattan distance is an alternative and considers that you can only connect points by moving horizontally or vertically and not diagonally as the Euclidean distance does. It uses a distance function similar to the movement over a grid (like the map of Manhattan, hence the name). An illustration of the difference is shown in Fig. 5.1.

The choice for either one of the two approaches highly depends on the dataset and can mostly only be determined after rigorous experimentation.

A generalization of the above metrics is the so-called Minkowski distance:

$$minkowski\_distance(q, x_i, x_j) = (\sum_{k=1}^{p}|x_i^k - x_j^k|^q)^{\frac{1}{q}} \tag{5.3}$$

We can see that $minkowski\_distance(1, x_i, x_j) \equiv manhattan\_distance(x_i, x_j)$ and $minkowski\_distance(2, x_i, x_j) \equiv euclidean\_distance(x_i, x_j)$. When considering these distance metrics, one should also consider whether scaling the data is needed or not.

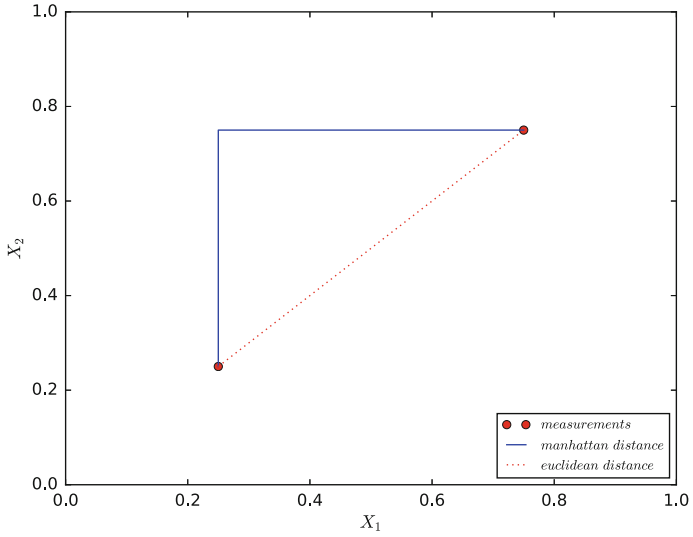

Fig. 5.1 Difference between Euclidean and Manhattan distance

Otherwise, certain attributes with a high magnitude and spread of observed values might get a very dominant role in the distance calculations.

When we have a combination of numerical and categorical attributes we cannot use the above distance metric. We could transform our categorical attributes into a binary representation, each value of the attribute becoming a new binary attribute. Another metric that can be used is Gower's similarity measure. With Gower's similarity, we only use an attribute k to measure the distance in case it has a value for both instances i and j (i.e. both are not unknown). The similarity for the attribute k is $s(x_i^k, x_j^k)$. It is defined in a different way for different types of variables. The outcome is always scaled to [0, 1]. For dichotomous variables (present or not) it is defined as

$$s(x_i^k, x_j^k) = \begin{cases} 1 & \text{when } x_i^k \text{ and } x_j^k \text{ are both present} \\ 0 & otherwise \end{cases} \tag{5.4}$$

For categorical data:

$$s(x_i^k, x_j^k) = \begin{cases} 1 & \text{when } x_i^k = x_j^k \\ 0 & otherwise \end{cases} \tag{5.5}$$

And for numerical data:

$$s(x_i^k, x_j^k) = 1 - \frac{|x_i^k - x_j^k|}{R_k} \text{ where } R_k \text{ is the range of } k \tag{5.6}$$

In order to compare to instances, we compute these functions for all attributes and divide the sum of these values by the number of possible comparisons. This leads us to Gower's similarity measure:

$$gowers\_similarity(x_i, x_j) = \frac{\sum_{k=1}^{p} s(x_i^k, x_j^k)}{\sum_{k=1}^{p} \delta(x_i^k, x_j^k)} \tag{5.7}$$

where $\delta(x_i^k, x_j^k)$ is defined as

$$\delta(x_i^k, x_j^k) = \begin{cases} 1 & \text{when } x_i^k \text{ and } x_i^k \text{ can be compared (i.e. both have values)} \\ 0 & otherwise \end{cases} \tag{5.8}$$

The distance is calculated by taking 1 minus the similarity. This concludes our discussion of distance metrics between individual data points.

5.2.2 Person Level Distance Metrics

When we want to cluster datasets of individuals, it becomes a bit more difficult. We now need a distance metric between complete datasets. To determine a suitable metric, we first need to understand how comparable datasets are. If datasets consist of time series that have been measured at the same granularity there are various ways to compute the distance between them. This does not hold for the general case where we do not assume a temporal ordering of the instances. Let us consider the case without temporal ordering first, and then look into comparing time series.

5.2.2.1 Non-temporal Distance Metrics

We have a set of variables $X_1, \ldots, X_p$. For each variable X_i, we have a number of measurements N_{qs_j} for person j: $x^i_{1,qs_j}, \ldots, x^i_{N_{qs_j},qs_j}$. For each person we summarize these values for the variable X_i in a single value, thereby creating a single instance per person. Using these instances we can apply the distance metrics we have explained previously. We can use the mean, standard deviation, minimum, maximum, or whatever summarizing function is deemed appropriate in the domain. For categorical attributes we can create new binary attributes for each value of the original attribute and use a similar approach. While our approach is simple (which is always nice), we might have lost a lot of information in our summarization step.

An alternative is to summarize the measurements for a specific variable of a person by means of its distribution (e.g. for the normal distribution this would be μ and σ^2). Hence, we fit a distribution to the data and use the parameter values of the model to summarize the values collected for a person and the difference in these values signifies the distance.

While the previous option is better than just summarizing by means of a single number, a third alternative exists, which is based on statistical tests. Specifically, we can test how the distributions of a variable for two different persons varies for numerical data, resulting in a probability for the assumption that both originate from the same distribution. This probability is often referred to as the p-value. The closer the distributions are, the smaller is the p-value. So we take $1-p$ as a distance metric. Any statistical test can be used depending on the distributions. The Kolmogorov-Smirnov test (cf. [73]) could be a nice choice here since it does not make any assumptions about the underlying distribution. A graphical illustration of the three approaches that have been discussed is shown in Fig. 5.2.

5.2.2.2 Temporal Distance Metrics

If we want to compare time series that are measured with the same granularity we can use other distance metrics. There are three options as discussed by Liao [81]: *raw data-based*, *feature-based*, and *model-based*.

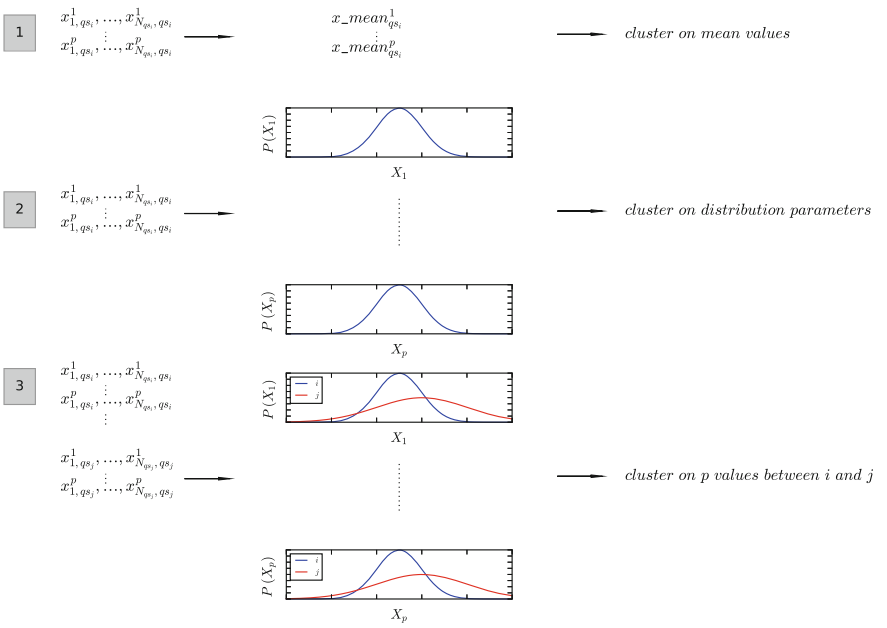

Fig. 5.2 Three approaches for calculating non-temporal person-level distances

Raw-Based Distance Metrics. Let us consider the *raw data-based* clustering first. For this approach we take the raw series of data for each attribute and define a distance metric. In Fig. 5.3, two series of accelerometer data generated by Arnold and his training buddy Eric are shown. We want to discuss three versions of the raw data-based approach. The first approach focuses on the differences between

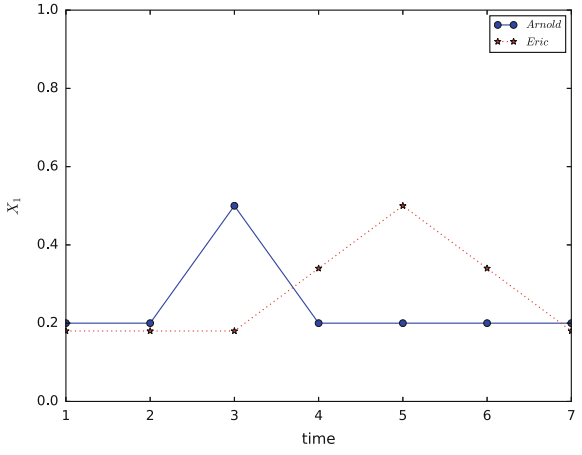

Fig. 5.3 Two example time series of Arnold and his buddy Eric

individual data points using for example the Euclidean distance. This is done by taking the vector with all measurements for a specific attribute l over time:

$$euclidean\_distance\_per\_attribute(x^l_{qs_i}, x^l_{qs_j}) = \sqrt{\sum_{k=1}^{N}(x^l_{k,qs_i} - x^l_{k,qs_j})^2} \tag{5.9}$$

Note that this does assume an equal number of points (i.e. we have the same N for both datasets). We can find the overall distance by summing the values over all attributes:

$$euclidean\_distance(x_{qs_i}, x_{qs_j}) = \sum_{l=1}^{p} euclidean\_distance\_per\_attribute(x^l_{qs_i}, x^l_{qs_j}) \tag{5.10}$$

An alternative is to consider the cross correlation between the two different time series. We can use the Pearson correlation coefficient, however the time series we compare might be shifted. We therefore use an approach to handle shifts in the patterns using a *lag*. Consider the two time series in Fig. 5.3 again. We see that they are very similar, only the peak in Eric's values has been shifted in time. Given a shift in time τ, the cross-correlation coefficient is defined as follows:

$$ccc(\tau, x^l_{qs_i}, x^l_{qs_j}) = \sum_{k=1}^{min(N_{qs_i}, N_{qs_j} - \tau)} x^l_{k,qs_i} \cdot x^l_{k+\tau,qs_j} \tag{5.11}$$

We see that the values of person j are shifted and the product of the values is taken. We sum over all available time points for which we can pair them up. The higher the value of this metric, the more the time series are aligned: if peaks align the product will become highest. Finding the value for τ which maximizes Eq. 5.11 is an optimization problem. In the end, the value for the distance can be defined as:

$$cc\_distance(x_{qs_i}, x_{qs_j}) = \underset{\tau=1,\ldots,min(N_{qs_i}, N_{qs_j})}{\text{argmin}} \sum_{k=1}^{p} \frac{1}{ccc(\tau, x^k_{qs_i}, x^k_{qs_j})} \tag{5.12}$$

We optimize the value of τ over all attributes as the time series should be shifted by the same value across all attributes. A third *raw data-based approach* is called *dynamic time warping (DTW)* (cf. [14]). The cross-correlation coefficient allows for time series that are shifted, but DTW can also take into account that there is a difference in speed between different time series. For instance, if we consider Arnold and Eric their sequences seem to align quite well, except that Eric slowly builds up towards his peak while Arnold does not. We might want to consider these series as relatively close to each other. The DTW algorithm tries to pair measurements (or time points/instances to phrase it differently) of the two time series, i.e. we match each time point in one series to a time point in the other series. These pairs are ordered and can be identified by an index k going from 1 to the time series with the longest

length (i.e. $max(N_{qs_i}, N_{qs_j})$). Our pairing does come with the constraints that the time order needs to be preserved (monotonicity condition) and we need to match the first and last points (boundary condition). Let $seq(k, qs_j)$ denote the sequence number of pair k with respect to qs_j, then we formalize the monotonicity constraint as follows:

$$\forall l \in 2, \ldots, max(N_{qs_i}, N_{qs_j}) : (seq(l, qs_i) \geq seq(l-1, qs_i)) \wedge (seq(l, qs_j) \geq seq(l-1, qs_j)) \tag{5.13}$$

Furthermore, the boundary condition is:

$$seq(1, qs_i) = seq(1, qs_j) = 1 \tag{5.14}$$

$$seq(max(N_{qs_i}, N_{qs_j}), qs_j) = N_{qs_j} \tag{5.15}$$

$$seq(max(N_{qs_i}, N_{qs_j}), qs_i) = N_{qs_i} \tag{5.16}$$

The problem of finding a matching is graphically illustrated in Fig. 5.4. We see the time series of Arnold on the left and the series of Eric at the bottom. Each square in the figure represents a possible pair. Each row is a time point for Arnold while each columns represents a time point for Eric. Moving ahead in time for Arnold is the same as moving up, while moving to the right is moving ahead in time for Eric. To find the pairs, we start at the bottom left and continue by matching points. To find a new pair, we can move up in this figure or move to the right, or both. Given our monotonicity constraint, we can never move down or to the left and our boundary condition requires that we start at the bottom left and end at the top right. The blue squares give an example of a path. The procedure to determine the minimum cost per position in the figure is shown below. In the end, the minimum cost in the upper right corner is returned as the dynamic time warping distance.

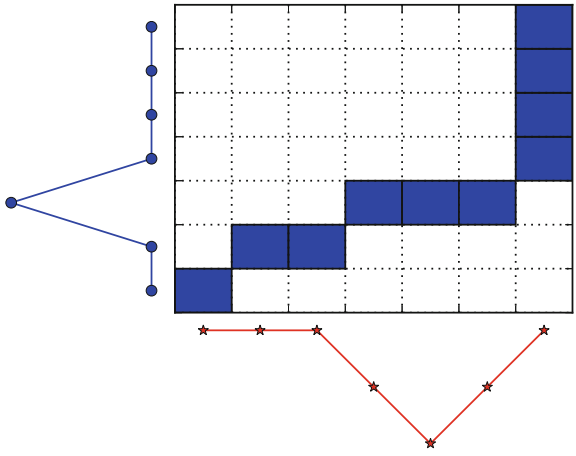

Fig. 5.4 Example dynamic time warping for example series of Arnold and Eric

Algorithm 3: Dynamic Time Warping

dtw_distance($\mathbf{X}^{\mathcal{T}}_{\mathbf{qs_i}}, \mathbf{X}^{\mathcal{T}}_{\mathbf{qs_j}}$) :
for $k \in 1, \ldots, N_{qs_i}$ **do**
  cheapest_path$(k, 0) = \infty$
end
for $k \in 1, \ldots, N_{qs_j}$ **do**
  cheapest_path$(0, k) = \infty$
end
cheapest_path(0,0) = 0
for $k = 1, \ldots, N_{qs_i}$ **do**
  for $l = 1, \ldots, N_{qs_j}$ **do**
    $d = \text{distance}(x_{k,qs_i}, x_{l,qs_j})$
    cheapest_path $= d + min(\{\text{cheapest\_path}(k-1, l), \text{cheapest\_path}(k, l-1), \text{cheapest\_path}(k-1, l-1)\})$
  end
end
return *cheapest_path*(N_{qs_i}, N_{qs_i})

The algorithm states that we cannot move outside of our time series (giving an infinite value for moving before the first point). We then calculate the minimum cost for each position in our search space (i.e. the squares shown in Fig. 5.4). This is the cost of the difference between the values in that square and the cheapest path that leads towards it. We typically use the Euclidean distance as the distance metric. Many improvements have been applied to this algorithm, for instance, limiting the maximum difference in time points of pairs, and more efficient calculations (e.g. the Keogh bound [70]). With this approach, we do not consider individual attributes but focus on the distance between the values of all attributes since we are trying to match time points over the whole dataset. According to some domain knowledge, it would also be possible to consider attributes on an individual basis and perform DTW individually if desired. Although we have not discussed categorical attributes in the context of raw data-based approach, they can be treated in the same way as we have mentioned before, by creating binary attributes per category.

Feature-Based Distance Metrics. As said before, we can also take a *feature-based* approach to comparing two time series. In order to do so, we extract features from the time series. For this purpose we can employ the same distance metrics as we have explained for the general case $\mathbf{X}$ by simply ignoring the temporal ordering. Alternatively, features similar to Chap. 4 can be used. To compare two time series we finally compare the derived features.

Model-Based Distance Metrics. For the *model-based* approach we fit a model our time series (e.g. a time series model as explained later in Chap. 8) and use the parameters of the model for the characterization of the time series. We compute the difference between these parameters and use it as the distance between persons.

5.3 Non-hierarchical Clustering

We have defined a number of distance metrics for both the cases of individual points and person level datasets. Given these distance measures we can now start clustering. For convenience we will use the notation for the individual data points.

We will start with the algorithm called *k-means clustering* [82]. In this approach, a predefined number of clusters k is found. Each cluster can be identified by a cluster center. The cluster centers are initially set randomly and then refined in a loop. The algorithm is shown in Algorithm 4.

Algorithm 4: k-means clustering

for $i = 1, \ldots, k$ **do**
  centers[k] = random point in the clustering space
end
prev_centers = []
while *prev_centers != centers* **do**
  prev_centers = centers
  cluster_assignment = []
  for $i = 1, \ldots, N$ **do**
    cluster_assignment[i] = $\text{argmin}_{j=1,\ldots,k}$ distance(x_i, centers[j])
  end
  for $j = 1, \ldots, k$ **do**
    centers[j] = $\frac{\sum_{l \in \{i | \text{cluster\_assignment[i]}=j\}} l}{|\{i | \text{cluster\_assignment[i]}=j\}|}$
  end
end

In the main algorithm loop we compute which cluster each point belongs to. This cluster is selected based on the minimum distance between the point and the cluster center. Once we have calculated this for all data points, we recompute the center of the cluster by taking the average over all data points in the cluster. This continues until the centers do not change anymore (or only change with a very small value). This approach is quite intuitive for individual data points and approaches where we aggregate datasets to single points.

Let us consider an example from the quantified self domain. Imagine we have data points that represent two dimensions of the accelerometer data and we want to cluster them in two clusters (i.e. $k = 2$). Figure 5.5 shows an example of the first steps of the k-means clustering algorithm. Here, the blue and red points are the data points while the black points are the cluster centers. After step 4, only one more update of the centers is required before the centers stabilize and the algorithm terminates. For distance metrics that are computed between whole datasets (i.e. the person level) and are raw-based k-means clustering is neither appropriate nor intuitive (what does the center of a set of datasets mean?). It has also been shown that k-means clustering does not always work well with dynamic time warping (cf. [90]).

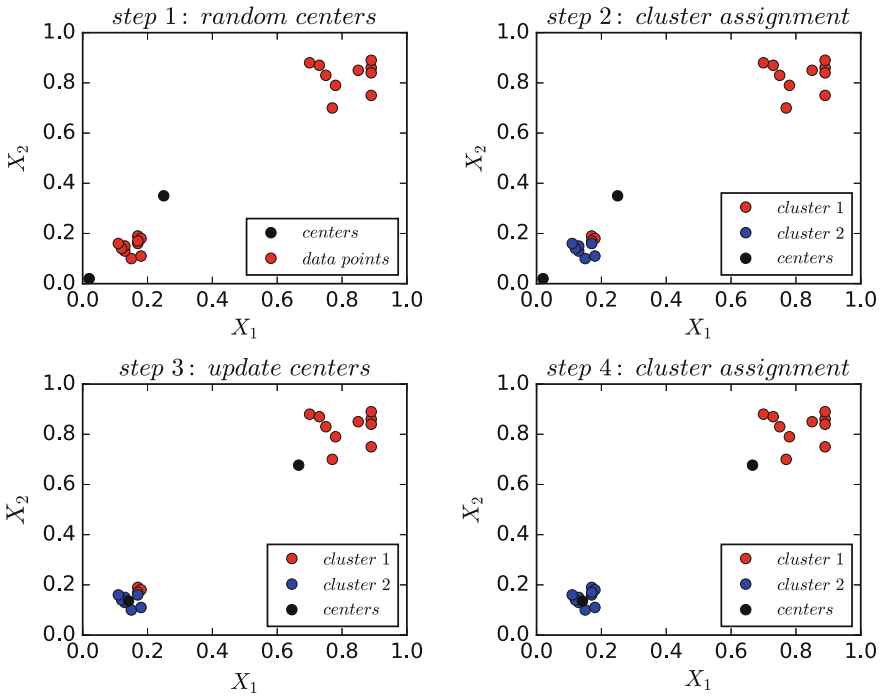

Fig. 5.5 Example of first steps of k-means clustering algorithm (with $k = 2$)

An alternative approach is k-medoids (cf. [67]). Instead of assigning cluster centers that are averages over the points belonging to the cluster, the k-medoids algorithm selects points from the dataset as cluster centers. This solves our previous problem we have identified with the person level data and also works well with dynamic time warping. The algorithm is very similar to the k-means clustering and is shown in Algorithm 5. The difference appears where the cluster center is assigned. Here, the actual point that minimizes the distance to all points in the cluster is selected.

Let us briefly take a step back: Without explicitly referring to clustering we have already seen another approach in Sect. 3.1.1.2 that supports clustering, that is Gaussian mixture models. In Sect. 3.1.1.2, we used this approach to identify outliers. In order to do so, we assumed that the data points were generated from a probability distribution that was constituted by a number of independent normal distributions and identify outliers with respect to this probability distribution. Implicitly we assumed that each data points, that is not an outlier, belongs to one of the independent normal distributions. Thus, each of these independent normal distributions can be interpreted as a cluster. What is the difference between mixture models and k-means/k-medoids clustering? For k-means/k-medoids we did not assume a probability distribution that generated the data points. Both approaches are appropriate for different applications: While mixture models are used for outlier detection or estimating probability densi-

Algorithm 5: k-medoids clustering

for $i = 1, \ldots, k$ **do**
  centers[k] = random point from $x_1, \ldots, x_N$ not part of centers yet
end
prev_centers = []
while *prev_centers != centers* **do**
  prev_centers = centers
  cluster_assignment = []
  for $i = 1, \ldots, N$ **do**
    cluster_assignment[i] = $\text{argmin}_{j=1,\ldots,k}$ distance(x_i, centers[j])
  end
  for $j = 1, \ldots, k$ **do**
    centers[j] = $\text{argmin}_{l \in \{i|\text{cluster\_assignment[i]}=j\}} \sum_{q \in \{i|\text{cluster\_assignment[i]}=j\}} \text{distance}(x_l, x_q)$
  end
end

ties, k-means clustering supports visualization and compression. By compression we mean, that high dimensional data points are represented by the cluster they belong to.

5.4 Hierarchical Clustering

The approaches we have seen so far require a predefined number of clusters and we only considered cluster that did not overlap. In hierarchical clustering, however, we drop both requirements. We either start with one big cluster and sequentially refine it (*divisive clustering*) or start with each instance in its own cluster and combine clusters. The latter is called *agglomerative clustering*. Before we dive into the details of the approach, let us consider the end-product of such a clustering which is often a *dendrogram*. The dendrogram shows the results of clustering on different levels, going from one cluster at the top to the most refined clusters at the bottom. This allows us to select the level of clustering which is appropriate for the domain. For an extensive overview of the approaches, see [68].

Figure 5.6 shows an example based on datasets of a number of quantified selves. Each depth of the dendrogram represents a cutoff value for our algorithms. Let us look into both approaches in a bit more detail.

5.4.1 Agglomerative Clustering

In agglomerative clustering, we start with each data point being an independent cluster and combine them until all data points constitute one cluster. Which clusters to combine depends on the distance between clusters. We assume to have a distance function between two data points x_i and x_j (which we have discussed extensively

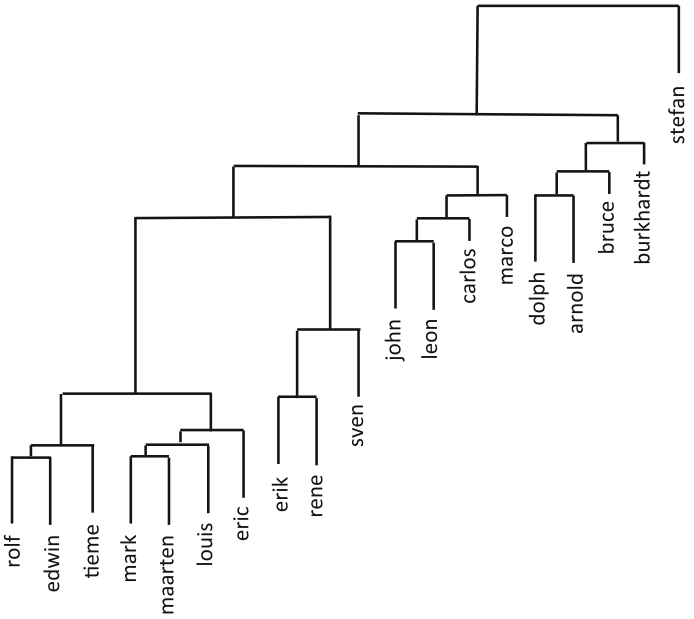

Fig. 5.6 Example of a dendrogram

before) noted as $distance(x_i, x_j)$. Given that we have clusters C_k and C_l we can define the distance between clusters in a variety of ways. We will discuss three here. First, the *single linkage* defines cluster distance as:

$$d_{SL}(C_k, C_l) = \min_{x_i \in C_k, x_j \in C_l} distance(x_i, x_j) \tag{5.17}$$

Hence, we take the distance between the two closest points as the distance between the clusters. The opposite is called *complete linkage* and takes the distance between two points that are farthest apart:

$$d_{CL}(C_k, C_l) = \max_{x_i \in C_k, x_j \in C_l} distance(x_i, x_j) \tag{5.18}$$

Third (and naturally) we can use the average distance between the points in the clusters referred to as the *group average*:

$$d_{GA}(C_k, C_l) = \frac{\sum_{x_i \in C_k} \sum_{x_j \in C_l} distance(x_i, x_j)}{|C_k| \cdot |C_l|} \tag{5.19}$$

A slightly different criterion is Ward's method [124]. It defines the distance between clusters as the increase in the standard deviation when the clusters are merged. Assume that m_{C_i} represents the center of cluster C_i, then the distance is defined as:

$$d_{Ward}(C_k, C_l) = \sum_{x_i \in C_k \cup C_l} ||x_i - m_{C_k \cup C_l}||^2 - \sum_{x_j \in C_k} ||x_j - m_{C_k}||^2 - \sum_{x_n \in C_l} ||x_n - m_{C_l}||^2 \tag{5.20}$$

Each one of the between cluster distance metrics comes with its own pros and cons, but in general the latter two provide a nice middle ground between a very strict condition for joining clusters (complete linkage) and a very loose one (single linkage). The algorithm to merge clusters is expressed in Algorithm 6. It works by means of a predefined threshold *th* for the maximum distance in order to still merge clusters. Intuitively, this threshold represents a horizontal position in the dendrogram, resulting in a certain division in clusters.

Algorithm 6: Agglomerative clustering

```
clusters = { }
for x_i ∈ X do
 | clusters = clusters + {x_i}
end
while True do
 | C_k, C_l = argmin_{C_k ∈ clusters, C_l ≠ C_k ∈ clusters} d(C_k, C_l)
 | if d(C_k, C_l) > th then
 |  | return clusters
 | else
 |  | clusters = clusters \ {C_k}
 |  | clusters = clusters \ {C_l}
 |  | clusters = clusters + {C_k + C_l}
end
```

We can see that we start with clusters of one data point and merge clusters until we can no longer find clusters that are sufficiently close.

5.4.2 Divisive Clustering

As said, divisive clustering works right in the opposite direction of agglomerative clustering. Therefore we start with a single cluster. Let us define the dissimilarity of a point to other points in its cluster:

$$dissimilarity(x_i, C) = \frac{\sum_{x_j \neq x_i \in C} distance(x_i, x_j)}{|C|} \tag{5.21}$$

Using this metric we can split a cluster C by considering the point with the greatest dissimilarity to the cluster. We create a new cluster C' for this and continue moving the most dissimilar points from cluster C to C'. We stop when there is no point left that is less dissimilar to the points in cluster C' than it is to the remaining points in cluster C. So what cluster should we select for the process we have just described? Various criteria have been defined, one being the cluster with the largest diameter (cf. [68]):

$$diameter(C) = \max_{x_i, x_j \in C} distance(x_i, x_j) \tag{5.22}$$

In other words, the diameter is the maximum distance between points in the cluster. This is used in Algorithm 7.

Algorithm 7: Divisive clustering

```
clusters = {{x_1, ..., x_N}}
while |clusters| < N do
    C = argmax_{C∈clusters} diameter(C)
    clusters = clusters \ C
    most_dissimilar_point = argmax_{x∈C} dissimilarity(x, C)
    C = C \ most_dissimilar_point
    C_new = {most_dissimilar_point}
    point_improvement = ∞
    while point_improvement > 0 do
        x = argmax_{x∈C} dissimilarity(x, C)
        distance_C = dissimilarity(x, C)
        distance_C_new = dissimilarity(x, C_new)
        point_improvement = distance_C_new - distance_C
        if point_improvement > 0 then
            C = C \ x
            C_new = C_new + x
        end
    end
    clusters = clusters + C + C_new
end
return clusters
```

Although we do not explicitly have the threshold set in this algorithm but work on a cluster by cluster basis, a dendrogram as we have seen before is the result of this procedure.

5.5 Subspace Clustering

While the approaches above are simple, intuitive, and in general work quite nicely, they do have some disadvantages when it comes to our quantified self setting. We might have a huge attribute space (we measure more and more around ourselves) and this causes several problems: (1) our approaches will take a long time to compute, (2) calculating distances over a large number of attributes can be problematic and distances might not distinguish cases very clearly, and (3) the results will not be very insightful due to the high dimensionality. Hence, we need to define a subset of the attributes (or subspace) to perform clustering. We could do this manually or use dimensionality reduction approaches such as Principal Component Analysis. However, the manual approach would need a lot of experimentation, while the latter would not provide intuitive results, as we transform the attributes that initially had a meaning to a less meaningful new space. Methods from *subspace clustering* come to rescue here. We will explain one specific subspace clustering algorithm, namely the *CLIQUE* algorithm [4].

Starting point for our explanation is that we create so-called *units* that partition the data space. Hereto, we split the range of each variable up into ϵ distinct intervals. This is exemplified for the dataset shown in Fig. 5.7. The splits are shown by the dotted lines.

Assuming $k \leq p$ dimensions or attributes (we can take subsets of the attributes), a unit u is defined by means of boundaries per dimension: $u = \{u_1, \ldots, u_k\}$.

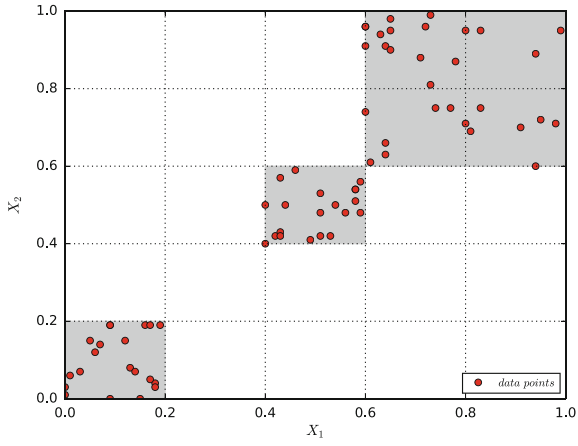

Fig. 5.7 Example of units in subspace clustering

A bound is provided for each dimension for the unit. $u_i(l)$ is the lower bound and $u_i(h)$ the upper bound for dimension i. A point belongs to a unit when it falls within all bounds:

$$belongs\_to(x, u) = \begin{cases} 1 & \forall_{i \in 1,\ldots,k} u_i(l) \leq x^i < u_i(h) \\ 0 & \text{otherwise} \end{cases} \tag{5.23}$$

For each unit we can define the *selectivity* and *density* of the unit u:

$$selectivity(u) = \frac{\sum_{i=1}^{N} belongs\_to(x_i, u)}{N} \tag{5.24}$$

$$dense(u) = \begin{cases} 1 & selectivity(u) > \tau \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

In other words, the selectivity is defined as the fraction of instances belonging to the unit u out of all instances, while a unit is called dense when the selectivity exceeds some threshold τ. In Fig. 5.7, we choose τ so that at least one instance is in the unit. This results in 6 dense units. These units are depicted in grey. As said, the units do not have to cover all dimensions (in fact we would rather like it if they do not). Assuming a unit covers k dimensions (with $k \leq p$) we define a cluster as a maximal set of connected dense units. Units $u_1 = \{r_1, \ldots, r_k\}$ and $u_2 = \{r'_1, \ldots, r'_k\}$ are directly connected when they have what is called a common face (i.e. when they share a border of a range on one dimension and have the same ranges on the others):

$$common\_face(u_1, u_2) = \begin{cases} 1 & \exists i \in 1, \ldots, k : (r_i(l) = r'_i(h) \vee r_i(h) = r'_i(l)) \wedge \\ & \forall j \neq i \in 1, \ldots, k : (r_i = r'_i) \\ 0 & \text{otherwise} \end{cases} \tag{5.26}$$

Furthermore, indirect connections are also considered. Overall units are connected in a cluster when:

$$connected(u_1, u_2) = \begin{cases} 1 & common\_face(u_1, u_2) \vee \\ & \exists u_3 : common\_face(u_1, u_3) \wedge common\_face(u_2, u_3) \\ 0 & \text{otherwise} \end{cases} \tag{5.27}$$

If we consider dense units as defined previously that are connected in our figure we obtain three clusters. These clusters can be specified by means of ranges of values that make up the region. For example, the upper right cluster can be expressed as $(0.6 \leq X_1 < 1) \wedge (0.6 \leq X_2 < 1)$ but also by $(0.6 \leq X_1 < 0.8) \wedge (0.8 \leq X_1 < 1) \wedge (0.6 \leq X_2 < 1)$. We call the *minimal description* of a cluster the smallest set of regions that still covers all units in the cluster. In our case this would be the first

description. Alright, the stage is set. How do we find units that are dense over all these different dimensions? The first problem we tackle is finding these units efficiently. Given the size of our search space, we cannot make calculations for each possible unit in each possible subset of our dimensions. To reduce the search space, we use the fact that a unit u can only be dense in k dimensions if all units of $k-1$ dimensions that are a subset of the constraints for unit u are also dense. This makes sense as the unit u covers a smaller part of the data space (it splits the data up in an additional dimension) and thus can never have more data instances in it. Thus we start with 1 dimension and work our way up to more dimensions. The generation of candidate units with k dimensions given the units with $k-1$ attributes known to be dense is expressed in Algorithm 8. Here, C_k denotes the set of candidate units of dimension k and R_{k-1} stands for the set of dense units of dimensions $k-1$. $u_i(a)$ refers to the name of the ith attribute of unit u. We look for two units that are part of the dense units in dimensions $k-1$, of which the attributes and bounds overlap in $k-2$ units and add the two non overlapping attributes (and associated ranges) to create a unit with dimension k, so we essentially create a unit with one additional attribute compared to the two units we have identified for dimension $k-1$. An ordering is assumed (indicated by $<$) to avoid doubles. For all candidates generated based on this algorithm, we compute whether they are dense or not.

Algorithm 8: Dense unit candidate generation from $k-1$ to k attributes

$C_k = []$
for $u, u' \in R_{k-1}$ **do**
  if $u_1(a) == u'_1(a) \wedge u_1(h) == u'_1(h) \wedge u_1(l) == u'_1(l) \wedge \cdots \wedge$
  $u_{k-2}(a) == u'_{k-2}(a) \wedge u_{k-2}(h) == u'_{k-2}(h) \wedge$
  $u_{k-2}(l) == u'_{k-2}(l) \wedge u_{k-1}(a) < u'_{k-1}(a)$ **then**
    $C_k = C_k + < u_1, \ldots, u_{k-1}, u'_{k-1} >$
  end
end
return C_k

While this already helps in terms of computation, we can further improve matters by focusing on subspaces (i.e. subsets of the attributes) that contain a significant proportion of the overall data points. Assuming subspaces $S_1, \ldots, S_n$ we compute the number of points that belong to the units that are part of the subspace and that are part of a dense unit (following our previous algorithm). We call this the *coverage*:

$$coverage(S_i) = \sum_{u \in S_i} (dense(u) \cdot \sum_{i=1}^{N} belongs\_to(x_i, u)) \tag{5.28}$$

Only subspaces with a large coverage will be selected. We therefore sort the subspaces according to their coverage score: $S_1, \ldots, S_n$ where S_1 has the highest coverage. We want to create a set of selected subspaces I and those we want to prune P. For

this purpose we select a point i in the ordered list at which we split it: $I = \{S_1, \ldots, S_i\}$ and $P = \{S_{i+1}, \ldots, S_n\}$. We choose i based on a heuristic that aims at minimizing the number of bits required to send information on the values of the coverage for all subspaces. The lower the amount of information we need to send, the better the split we have obtained. Given split i the minimum information we are required to send is the average coverage of the subspaces in I (called $\mu_I(i)$) and P ($\mu_P(i)$) and the deviation of the coverage of each subspace from this average. Since we send the information in bits we take the $\log_2$ of these values:

$$\mu_I(i) = \frac{\sum_{j=1}^{i} coverage(S_j)}{i} \tag{5.29}$$

$$\mu_P(i) = \frac{\sum_{j=i+1}^{n} coverage(S_j)}{(n-i)} \tag{5.30}$$

$$information(i) = \log_2(\mu_I(i)) + \log_2(\sum_{j=1}^{i} |coverage(S_j) - \mu_I(i)|) + \log_2(\mu_P(i)) + \log_2(\sum_{j=i+1}^{n} |coverage(S_j) - \mu_P(i)|) \tag{5.31}$$

All we need to do is find the value of i that minimizes this sum. The idea behind this approach is that the pruned set will contain all subspaces with very low coverages (and thus low variation and information).

We now have a selection of subspaces of certain dimensions and we know how to effectively compute dense units within those subspaces. A logical next step is to find units that when combined make up clusters in a certain subspace. We can do this using a depth first search like algorithm. Part of the procedure is shown in Algorithm 9. We start with a cluster number n and a unit u. We then go through the k dimensions of the current subspace and look for neighbors on the left and right of the point to see whether they are dense as well and not part of a cluster yet. If this is the case, we continue along that avenue. This results in the assignment of units to cluster n. Once we have found the clusters, we describe them in terms of their ranges of values (as we previously indicated) and find a minimal way to do so. Agrawal et al. [4] discuss in detail how this can be done in an efficient way.

After all these steps have been performed we end up with a selection of suitable subspaces and a description of the clusters in those subspaces.

5.6 Datastream Clustering

Although subspace clustering helps us solve the dimensionality issue, some challenges and restrictions remain. All of the algorithms we have seen so far make some strong assumptions [1]:

Algorithm 9: Cluster generation

```
find_neighbors(u, n) :
cluster(u) = n
for j = 1, ..., k do
    u^l = u
    u^l_j(h) = u_j(l)
    u^l_j(l) = u_j(l) − 1
    if dense(u^l) ∧ cluster(u^l) = unknown then
        find_neighbors(u^l, n)
    end
    u^r = u
    u^l_j(l) = u_j(h)
    u^l_j(h) = u_j(h) + 1
    if dense(u^h) ∧ cluster(u^h) = unknown then
        find_neighbors(u^l, n)
    end
end
```

- They assume that an unlimited amount of data can be stored, such that multiple passes can be performed, e.g. in the k-means clustering. In our setting of the quantified self, storing all sensory data in a highly fine-grained manner might exceed the storing capacity of our mobile devices. A central repository would be an option, however, imagine Arnold's accelerometer data with more than 100 samples per second. Uploading this data would take way too much bandwidth.
- Another assumptions is that all data should be treated in the same way. However, the underlying mechanisms generating the data might evolve over time, requiring models to become outdated. For instance, Bruce might improve his ability to control his blood glucose level. This is referred to as *temporal locality* or *concept drift*. Note that this is different from our temporal/non-temporal predictors we have seen in the previous section.

Within the domain of *data stream mining* algorithms are being developed that no longer build on the assumptions above. We will give a few example for clustering approaches which tackle some of the problems listed earlier. One approach is to maintain a window of a particular size n and cluster only on the last n elements. Each new arriving instance then replaces an element in our window (e.g. the oldest) or we replace elements only with a certain probability less than 1 to avoid having to run the algorithms repeatedly.

An alternative approach is to store cluster centers for chunks of data and continue abstracting over these centers when our dataset grows [55]. This might sound a bit vague, but it is actually pretty straightforward. Imagine that we select the first m elements in our data. We cluster these m elements by means of k-medoids and identify k instances as centers. We assign a weight w to each center based on the number of instances that are part of the cluster. We continue this process with the next chunk of data of size m until we have done this $\frac{m}{k}$ times. We now have a new

dataset of m centers and their associated weights. We cluster these centers into k clusters again based on the weights of the medoids we had previously assigned. We can continue this process for multiple levels, and we only store the medoids, weights for each level. Hence, we greatly reduce our need for storing lots of data.

Another alternative is to use the mixture of normal models that we have discussed earlier. For this purpose, you can imagine that part of the data, for example the last n data points, is used to estimate an initial mixture of normals. This results in a compressed representation of the n data points in terms of the means and standard deviations of the mixture components. As new data comes in, the density estimates are updated. It has been shown that this is a very efficient way to handle data stream clustering [109].

5.7 Performance Evaluation

Clustering is not as clear cut in terms of performance metrics compared to supervised learning approaches which we will discuss in the next three chapters. So how do we know that we have found a good clustering? Well, there are some metrics that can help out. We will discuss one of the most prominent examples here, namely the *silhouette score* [101]. This cannot only help to evaluate the clustering but also to find good values for the parameters of various clustering approaches (e.g. k for k-means and k-medoids). We start by defining the average distance of a point to the other points in its cluster:

$$a(x_i) = \frac{\sum_{\forall x_j \in C_l} distance(x_i, x_j)}{|C_l|} \text{ where } x_i \in C_l \tag{5.32}$$

In addition, the silhouette score uses the average distance to the points in the cluster closest by

$$b(x_i) = \min_{\forall C_m \neq C_l} \frac{\sum_{\forall x_j \in C_m} distance(x_i, x_j)}{|C_m|} \text{ where } x_i \in C_l \tag{5.33}$$

We can now define the silhouette:

$$silhouette = \frac{\sum_{i=1}^{N} \frac{b(x_i) - a(x_i)}{max(a(x_i), b(x_i))}}{N} \tag{5.34}$$

It compares the distances $a(x_i)$ and $b(x_i)$ and divides it by the maximum of the two. Hence, it provides a measure on how tight the clusters themselves are relative to the distance to the clusters closest to them. The score can range from -1 to 1, where -1 is clearly the worst score one could obtain as apparently the distances from points to

other clusters are lower than those between points within the cluster. The closer to 1 the better it is, since close to 1 represents a low value for the $a(x_i)$'s (tight clustering) and high values for $b(x_i)$'s (cluster are far apart).

5.8 Case Study

Let us go back to our crowdsignals dataset. While our goal is not explicitly set to finding interesting clusters, clustering could still contribute to solving our problem (predict the label or the heart rate): if we can cluster our measurements in such a way that the membership of a cluster is predictive for the target, it would be a great contribution, making the presence of an instance in a cluster a new attribute. In addition, it provides us with insights into our data that will help to make choices in the next chapters. We will try several of the clustering approaches we have discussed in this chapter. While we have ample options to choose from in terms of distance metrics and a learning setup, we take a rather straightforward one. We will use the Euclidean distance as a distance metric (it is easy to understand, applicable to both k-means and k-medoids, and fast). For the selection of the learning setup we do not have much choice: we just have the dataset of a single quantified self. Therefore we aim to cluster instances in our data instead of selecting the person level. For the clustering we will focus on the accelerometer of the phone, i.e. the attributes *acc_phone_x*, *acc_phone_y*, and *acc_phone_z*. Our task is to find clusters that might be indicative for the type of activity being conducted (though we do not use the label information to generate the clusters). Of course, this selection of measurements is a bit arbitrary, but it is a set of measurements that is representative for most of our measurements.

5.8.1 Non-hierarchical Clustering

First, let us dive into non-hierarchical approaches, namely k-means and k-medoids.

5.8.1.1 K-Means

Given our dataset, we need to figure out the best setting for the number of clusters (k) first. For this, we run the algorithm with different values for k (ranging from 2 to 9 clusters) and measure the silhouette to judge the quality of the clustering. The result is shown in Fig. 5.8. Note that we only do some exploratory runs with a limited number of random initializations per setting to get an idea on the best value for k. From the figure, we can see that a value of $k = 6$ results in the highest score on the silhouette and the score is quite reasonable (0.743). Let us visually explore the clusters resulting

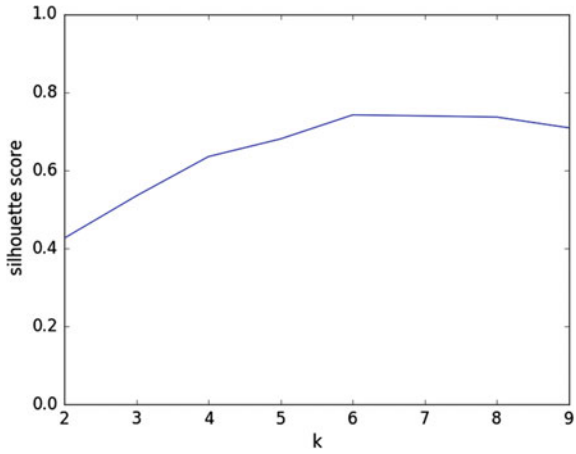

Fig. 5.8 Silhouette score of k-means for different values of k

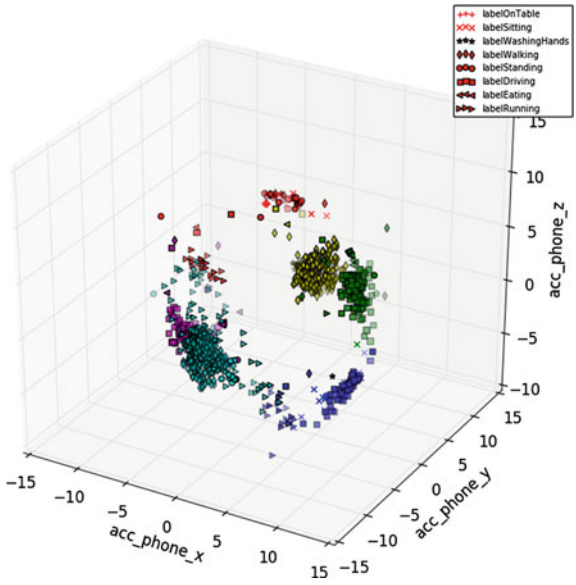

Fig. 5.9 Visualization of the clusters found with k-means with $k = 6$ (colors) and the labels (markers)

from this setting. We have illustrated them in Fig. 5.9. We also included the labels in the figure. Upon inspection we see quite a nice consistent clustering.

When considering the labels, it does seem that labels are not just randomly spread across the different clusters, for example the yellow cluster seems to contain a lot of walking label points. Table 5.1 shows more statistics about the clusters. We can see that each cluster has its own "niche" in terms of the different accelerometer measurements and we also observe the spread of labels across the clusters. For example, the instances with the phone lying on the table are nearly entirely covered by the first cluster, while the sitting behavior is caught in the third cluster. Walking,

Table 5.1 Distribution of measurements and labels over for k-mean clustering. Note that the percentage for the label indicates the percentage of total rows among which the label has been assigned

Attribute	Statistic	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Accelerometer data							
acc_phone_x	Mean	−0.36	8.24	8.00	−0.75	−8.21	−0.56
	Std	1.15	0.96	1.02	1.71	0.94	1.21
acc_phone_y	Mean	0.98	0.61	−2.35	−9.61	2.25	9.55
	Std	1.92	1.36	2.06	1.22	1.94	1.12
acc_phone_z	Mean	9.19	4.54	−4.80	0.23	−4.67	−0.56
	Std	1.06	1.36	1.08	1.49	1.44	1.62
Labels							
labelOnTable	Percentage (%)	99.56	0.44	0.00	0.00	0.00	0.00
labelSitting	Percentage (%)	2.40	0.40	97.20	0.00	0.00	0.00
labelWashingHands	Percentage (%)	7.02	1.75	1.75	56.14	0.00	33.33
labelWalking	Percentage (%)	1.87	0.94	0.47	46.14	0.47	50.12
labelStanding	Percentage (%)	4.74	1.42	0.00	48.34	0.47	45.02
labelDriving	Percentage (%)	1.67	55.56	20.00	0.00	22.22	0.56
labelEating	Percentage (%)	2.54	39.59	0.00	0.00	57.36	0.51
labelRunning	Percentage (%)	17.43	0.92	17.43	64.22	0.00	0.00

standing, and washing hands seem to occur equally frequent among clusters four and six. Having said that, we can expect that this clustering should be able to help us with our tasks.

As a final analysis, we plot the silhouette for the clusters that result per data point, this gives a nice indication on the quality of the various clusters, see Fig. 5.10. We see quite a consistent picture across the clusters.

5.8.1.2 K-Medoids

We have applied the k-medoids algorithm to the same problem, and study the same characteristics. In terms of the silhouette scores over different values for k we see a similar result as we have found for k-means: $k = 6$ is best again (Fig. 5.11). The best silhouette score is 0.742, similar to the one we previously obtained for k-means. The clusters are also pretty similar and so are the silhouette scores of the individual points. We therefore only show the table with the statistics, see Table 5.2. The table also does not indicate any significant differences.

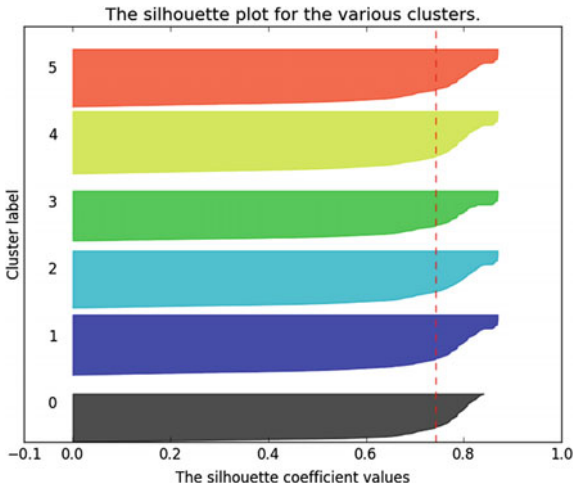

Fig. 5.10 Silhouette score of the data points in the different clusters with $k = 6$

Table 5.2 Distribution of measurements and labels over clusters for k-medoids clustering. Note that the percentage for the label indicates the percentage of total rows among which the labels has been assigned

Attribute	Statistic	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Accelerometer data							
acc_phone_x	Mean	−0.31	8.24	7.97	−0.86	−8.21	−0.56
	Std	1.06	0.96	1.01	1.62	0.94	1.21
acc_phone_y	Mean	1.03	0.59	−2.46	−9.52	2.25	9.55
	Std	1.89	1.40	2.25	1.46	1.94	1.12
acc_phone_z	Mean	9.26	4.53	−4.76	0.30	−4.67	−0.56
	Std	0.87	1.38	1.08	1.54	1.44	1.62
Labels							
labelOnTable	Percentage (%)	99.56	0.44	0.00	0.00	0.00	0.00
labelSitting	Percentage (%)	2.40	0.40	97.20	0.00	0.00	0.00
labelWashingHands	Percentage (%)	7.02	1.75	1.75	56.14	0.00	33.33
labelWalking	Percentage (%)	1.87	0.94	0.47	46.14	0.47	50.12
labelStanding	Percentage (%)	4.74	1.42	0.00	48.34	0.47	45.02
labelDriving	Percentage (%)	1.67	55.56	20.00	0.00	22.22	0.56
labelEating	Percentage (%)	2.54	39.59	0.00	0.00	57.36	0.51
labelRunning	Percentage (%)	8.26	0.92	21.10	69.72	0.00	0.00

5.8.2 Hierarchical Clustering

Finally, we are going to try a form of hierarchical clustering, namely the agglomerative clustering approach. While we no longer need a pre-defined number of clusters, we can select the number of clusters by choosing a certain point in the dendrogram. We use the Ward linkage function. Figure 5.12 shows the dendrogram for our problem at hand. When we create clusters, the highest silhouette score we obtain is 0.730 for $k = 6$.

Based on the silhouette (which hardly differs per approach) we arbitrarily select the k-means clustering approach to add a feature which represents the attribution of an instance in the dataset to a cluster.

5.9 Exercises

5.9.1 Pen and Paper

1. We have presented a number of person level distance metrics. While we did not discuss it explicitly, each one comes with their own pros and cons. Present an advantage and a disadvantage for each of the metric (hint: think of computational complexity, the amount of information taken into account, etc.).
2. Let us consider one of the person level distance metrics, namely the dynamic time warping. As we explained the approach, we tried to find the shortest path to match up all of our data points. This would be the distance between the attribute value of two persons. From the literature, it is known that the shortest path is not always the best option to use as a distance metric. Give an example that supports

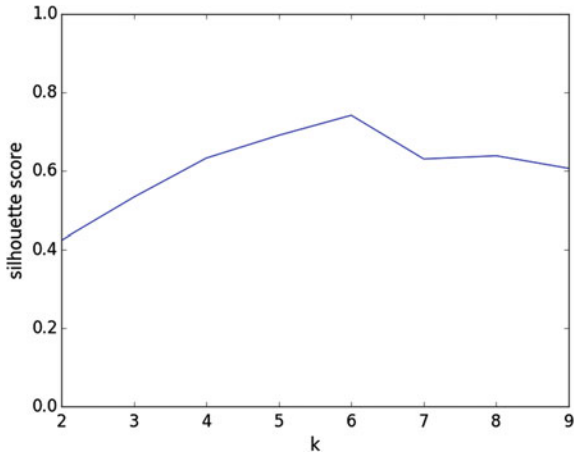

Fig. 5.11 Silhouette score of k-medoids for different values of k

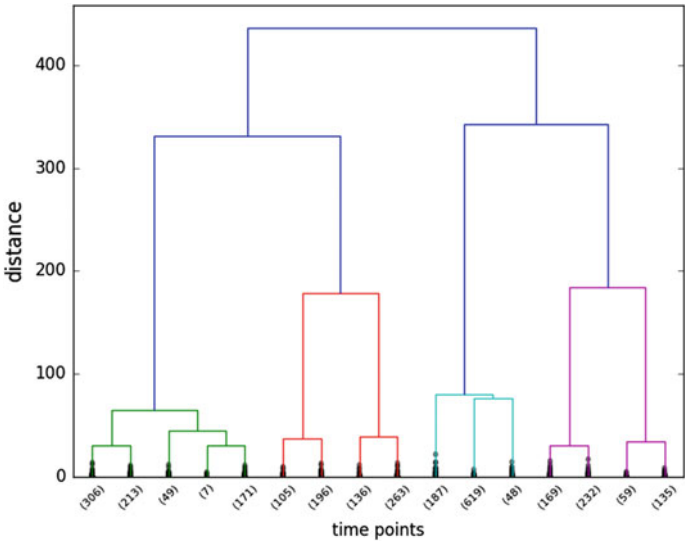

Fig. 5.12 Dendrogram for the crowdsignals dataset. Note that the numbers between brackets at the *bottom* of the dendrogram represent the number of instances

this statement. Furthermore, give an alternative dynamic time warping distance metric that avoids this disadvantage.

3. We have explained the k-mean and k-medoids algorithms. We did not talk about the guarantees that are provided on the quality of the solution though. Do you think we are guaranteed to find the optimal clustering in either k-means and k-medoids clustering? Explain why (not).
4. The literature suggests that it is very hard to use k-means clustering in combination with some of the person level distance metrics. Which one of the distance metrics would be hardest to use in combination with k-means? And why?
5. What is the computational complexity of both the k-means and the k-medoids algorithm?
6. In agglomerative clustering, Ward's criterion can result in a very different dendrograms compared to the other criteria such as the single and complete linkage criteria. What would you expect to be different if you compare the outcomes of the two approaches? And what could be the reason for this?
7. We have not used a subspace clustering approach in our case study but consider it to be a very relevant approach for the quantified self domain. Provide a concrete example (in terms of a dataset with certain features and certain distributions of the values of those features) where subspace clustering would be a far better choice compared to the more common clustering approach.
8. We have seen one metric to evaluate the quality of clustering namely the silhouette. Provide at least one additional metric for clustering quality and explain how it is computed.

5.9.2 Coding

1. We have focused on the phone's accelerometer data in our clustering, but did not touch upon the other sensors. Cluster the gyroscope data for the crowdsignals dataset using k-means, k-medoids, and hierarchical clustering. Do you see a similar clustering as we have seen for the accelerometer data? And how do the clusters relate to the activity?
2. Let us move on to your own dataset. Select a few relevant features from your own dataset and cluster them using one of the clustering approaches. Write down and illustrate your results.
3. Select either your own dataset, or the crowdsignals dataset. Compare different criteria for the agglomerative clustering and visualize the differences. Explain how the criteria influence the clustering and the shape of the resulting dendrogram.
4. Take the dataset covering multiple persons you have used in previous chapters. Apply k-medoids clustering with all the different person level distance metrics that have been discussed in this chapter. Show the results using these metrics and compare the results of the clustering for each metric.
5. Apply hierarchical clustering to the dataset covering multiple persons. Use only one person level metric (you can select which one). Compare the outcome to the k-medoids clustering result with the same person level metric.