Machine Learning for the Quantified Self

Chapter 5
Clustering
Overview

• Previously: we identified features
• Today: let us use the features to cluster
  – Learning setup
  – Distance functions
  – Clustering algorithms
Learning setup (1)

• Per instance:

\[
X = \begin{bmatrix}
  x_{1,qs1} \\
  \vdots \\
  x_{N,qs1} \\
  x_{1,qs_n} \\
  \vdots \\
  x_{N,qs_n}
\end{bmatrix}
\]
Learning setup (2)

- Per person:

\[ X = \begin{bmatrix} X_{q1} \\ \vdots \\ X_{qn} \end{bmatrix} \]
Distance metrics (1)

• Need different distance metrics per scenario

• Individual distance metrics (between an instance \( x_i \) and \( x_j \) over all attributes \( p \)):
  
  – Euclidean distance
    
    \[
    \text{euclidian\_distance}(x_i, x_j) = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2}
    \]

  – Manhattan distance
    
    \[
    \text{manhattan\_distance}(x_i, x_j) = \sum_{k=1}^{p} |x_{ik} - x_{jk}|
    \]
Individual distance metrics (1)

- The difference:

\[
\text{euclidian distance} \quad (x_i, x_j) = \sqrt{\sum_{k=1}^{p} (x_{ki} - x_{kj})^2} \quad (5.1)
\]

\[
\text{manhattan distance} \quad (x_i, x_j) = \sum_{k=1}^{p} |x_{ki} - x_{kj}| \quad (5.2)
\]

This can just be interpreted as the distance between the two points in the Euclidian space. The Manhattan distance is an alternative and considers that you can only connect points by moving horizontally or vertically, not diagonally as the Euclidian 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 Figure 5.1.

Fig. 5.1: Difference between Euclidian and Manhattan distance

The choice for one of the two is highly dependent on the dataset used and can mostly only be determined after rigorous experimentation. One can even derive a generalization of both metrics, called the Minkowski distance:

\[
\text{minkowski distance} \quad (q, x_i, x_j) = \left( \sum_{k=1}^{p} |x_{ki} - x_{kj}|^q \right)^{1/q} \quad (5.3)
\]

We can see that

\[
\text{minkowski distance} \quad (1, x_i, x_j) \implies \text{manhattan distance} \quad (x_i, x_j)
\]

and

\[
\text{minkowski distance} \quad (2, x_i, x_j) \implies \text{euclidean distance} \quad (x_i, x_j)
\]

When considering these distance metrics, one should also consider whether scaling of the data is needed or not. Otherwise, certain attributes with a high magnitude and spread of observed values might get a very dominant role in the distance calculations.
Individual distance metrics (2)

• Generalized form:

\[ minkowski\_distance(q, x_i, x_j) = \left( \sum_{k=1}^{p} |x_i^k - x_j^k|^q \right)^{\frac{1}{q}} \]

• Important to consider: scaling of data
• These all assume numeric values, what if we have other types?
  – We can use Gower’s similarity
Individual distance metrics (3)

- Gower’s similarity for different types of features:
  - Dichotomous attributes (present or not)
    \[ s(x^k_i, x^k_j) = \begin{cases} 
    1 & \text{when } x^k_i \text{ and } x^k_j \text{ are both present} \\
    0 & \text{otherwise} 
    \end{cases} \]
  - Categorical attributes:
    \[ s(x^k_i, x^k_j) = \begin{cases} 
    1 & \text{when } x^k_i = x^k_j \\
    0 & \text{otherwise} 
    \end{cases} \]
  - Numerical attributes:
    \[ s(x^k_i, x^k_j) = 1 - \frac{|x^k_i - x^k_j|}{R_k} \text{ where } R_k \text{ is the range of } k \]
Individual distance metrics (4)

• And we computer the final similarity as:

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

$$ \delta(x^k_i, x^k_j) = \begin{cases} 
1 & \text{when } x^k_i \text{ and } x^k_j \text{ can be compared (i.e. both have values)} \\
0 & \text{otherwise} 
\end{cases} $$
Distance metrics (2)

• What if we need to compare *datasets* instead of individual instance?
• We can identify person level distance metrics
• These are available in different forms:
  – For datasets without an explicit ordering
  – For datasets with a temporal ordering
Non-temporal personal level distance metrics (1)

- We have a number of options:
  1. Summarize values per attribute \(i\) over the entire dataset into a single value
     - e.g., mean, minimum, maximum, ….
     - use the same distance metrics we have seen before
  2. Estimate parameters of distribution per attributes \(i\) and compare those (e.g. \(N(\mu,\sigma^2)\))
     - e.g. normal distribution \(N(\mu,\sigma^2)\)
     - use the same distance metrics we have seen before upon the found parameter values
  3. Compare the distributions of values for an attribute \(i\) with a statistical test
     - e.g. use a Kolmogorov Smirnov test, resulting in a \(p\)-value
     - Take \(1-p\) as a distance metric
Non-temporal personal level distance metrics (2)

1. \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
1. \( x_{mean_1} \) \\
   \( x_{mean} \) \\
   \( cluster \ on \ mean \ values \)

2. \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
2. \( cluster \ on \ distribution \ parameters \)

3. \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
   \( x_1, q_{a_1}, \ldots, x_{N_{a_1}, q_{a_1}} \) \\
3. \( cluster \ on \ p \ values \ between \ i \ and \ j \)

Fig. 5.2: Three approaches for calculating person-level distances

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

The first raw-based approach we will discuss to compare the time-series is to look at the differences between the individual datapoints we have obtained, e.g. the Euclidian distance. This is performed in a similar way as we have expressed before except that we take a vector with the measurements of the values of a single attribute over time:
Temporal personal level distance metrics (1)

• What if we do have an ordering?
  – Raw-data based
  – Feature-based
  – Model-based
Raw-data based distances (1)

• Simplest case: assume an equal number of points and compute the Euclidean distance on a per point basis:

\[
\text{euclidian\_distance\_per\_attribute}(x_{qs_i}^l, x_{qs_j}^l) = \sqrt{\sum_{k=1}^{N} (x_{k,qs_i}^l - x_{k,qs_j}^l)^2}
\]

\[
\text{euclidian\_distance}(x_{qs_i}, x_{qs_j}) = \sum_{k=1}^{p} \text{euclidian\_distance\_per\_attribute}(x_{qs_i}^k, x_{qs_j}^k)
\]
Raw-data based distances (2)

• What if the time series are more or less the same, but shifted in time?
• We can use a lag to compensate, we denote the amount of time shifted by $\tau$
• The cross correlation coefficient between two dataset is computed by:

$$ccc(\tau, x^l_{qs_i}, x^l_{qs_j}) = \sum_{k=-\infty}^{\infty} x^l_{k,qs_i} \cdot x^l_{k+\tau,qs_j}$$
Raw-data based distances (3)

- What is the best value $\tau$ to shift?
- This is an optimization problem (note: one shift over all attributes):

$$cc_{distance}(x_{qs_i}, x_{qs_j}) = \arg\min_{\tau=1,...,\min(N_{qs_i},N_{qs_j})} \sum_{k=1}^{p} \frac{1}{cc(	au, x^k_{qs_i}, x^k_{qs_j})}$$
Next to this, we can also think about different frequencies at which different persons perform their activities.

We can use *dynamic time warping* for this purpose.

We try to make the best pairs of instances in sequences to find the minimum distance.
Dynamic Time Warping (1)

• Pairing?

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

The problem of finding a matching is graphically illustrated in Figure 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 column 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...
Dynamic Time Warping (2)

• Requirements of pairing
  – Time order should be preserved
    • monotonicity condition
  – First and last points should be matched
    • boundary condition
  – assume seq(k, q_{sj}) to represent the sequence number (in terms of instance ) of pair k with respect to the dataset q_{sj}
Dynamic Time Warping (3)

• Requirements of pairing
  – monotonicity condition
    \[ \forall l \in 2, \ldots, \max(N_{qs_i}, N_{qs_j}) : (seq(l, qs_i) \geq seq(l - 1, qs_i)) \land \\
    (seq(l, qs_j) \geq seq(l - 1, qs_j)) \]
  – boundary condition
    \[ seq(1, qs_i) = seq(1, qs_j) = 1 \]
    \[ seq(\max(N_{qs_i}, N_{qs_j}), qs_j) = N_{qs_j} \]
    \[ seq(\max(N_{qs_i}, N_{qs_j}), qs_i) = N_{qs_i} \]
Dynamic Time Warping (4)

Let \( \text{seq}(k, qs_j) \) denote the sequence number of pair \( k \) with respect to \( qs_j \), then we formalize the monotonicity constraint as follows:

\[
\text{seq}(l, qs_i) = \text{seq}(l_1, qs_i) \quad (5.13)
\]

Furthermore, the boundary condition is:

\[
\text{seq}(1, qs_i) = \text{seq}(1, qs_j) = 1 \quad (5.14)
\]

\[
\text{seq}(\max(N_{qs_i}, N_{qs_j}), qs_j) = N_{qs_j} \quad (5.15)
\]

\[
\text{seq}(\max(N_{qs_i}, N_{qs_j}), qs_i) = N_{qs_i} \quad (5.16)
\]

Fig. 5.4: Example dynamic time warping for example series of Arnold and Eric. The problem of finding a matching is graphically illustrated in Figure 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 boundary condition condition monotonicity condition condition.
Dynamic Time Warping (5)

• How do we find the best path?
  – We start at (0,0) and move up and to the right
  – Per pair we compute the cheapest way to get there given our constraints and the distance between the instances in the pair
  – As distance between instance we can for example use the Euclidean distance
  – Let us look at the algorithm.....
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 for each position in our search space (i.e. the squares shown in Figure 5.4) what the minimum cost is. 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 Euclidian distance as the distance function. Many improvements have been made to this algorithm, for instance limiting the maximum difference in time points of pairs, and more efficient calculations (e.g. the Keogh bound [65]). 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. Of course it would be possible to consider attributes on an individual basis and perform DTW on them individually if this is desired according to the domain knowledge available. Though we have not touched much upon categorial attributes, they can be treated in the same way as we have mentioned before: by creating binary attributes per category.

As we said before, we can also take a feature-based approach opposed to the raw-based approach. We extract features from our time series and for these features we can use the same distance metrics as we have explained for the general case. The extraction of features can be the same as we have seen for $X^T$, basically ignoring the ordering. Alternatively, features similar to Chapter 4 can be used and compared. For the model-based approach we fit a model towards our time series (e.g. a time series model as explained later in Chapter 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.
Dynamic Time Warping (7)

• Computationally expensive
  – Improvement: Keogh bound

• Would you be willing to allow for all matches provided they meet the conditions?

• Would the best match always be the best distance metric?
Temporal personal level distance metrics (2)

• What if we do have an ordering?
  – Raw-data based v
  – Feature-based
    • Same as non-temporal case we have seen
  – Model-based
    • Fit a time series model and use those parameters
      (again in line with the non-temporal case, except
      for the type of model being different)
Clustering approaches

• Ok, we know how to define a distance, now what clustering approaches do we have?
• Assume you do have some basic understanding
• Will scan through the options briefly
  – k-means
  – k-medoids
  – 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** \cite{75}. In the approach a predefined number of clusters $k$ is found. Each cluster is defined by means of a cluster center. The cluster centers are initially set randomly after which they are 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 = centers
while prev_centers != centers do
    prev_centers = centers
    cluster_assignment = []
    for $i = 1, \ldots, N$ do
        cluster_assignment[i] = argmin$_{j = 1, \ldots, k}$ distance($x_i$, centers[j])
    end
    for $j = 1, \ldots, k$ do
        cluster_center[j] = $\frac{\sum_{l \in \{i \mid \text{cluster_assignment}[i] = j\}} l}{|\{i \mid \text{cluster_assignment}[i] = j\}|}$
    end
end
```

In the main algorithm loop we compute for each point what cluster it 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 our 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 running the k-means clustering algorithm. Here, the big shapes are the data points while the smaller points are the cluster centers. After step 4 just 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) k-means clustering is not very usable and intuitive as the center represents a dataset in the center of the datasets assigned to the cluster. It has also been shown that k-means clustering does not always work well with dynamic time warping (cf. \cite{81}).
5.4 Hierarchical Clustering

The approaches we have seen so far require a predefined number of clusters and we only considered clusters that did not overlap. In hierarchical clustering, we drop both requirements. We either start with one big cluster and refine it more and more (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 mostly a dendrogram. The dendrogram shows the results of clustering on different levels.
Performance metric

• How do we select the best value for $k$?
• How do we evaluate whether a clustering is good?
• Silhouette can be used ([-1, 1], 1 is best)

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

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

$$b(x_i) = \min_{\forall C_m \neq C_l} \frac{\sum_{\forall j \in C_m} \text{distance}(x_i, x_j)}{|C_m|} \quad \text{where } x_i \in C_l$$
CrowdSignals accelerometer data
5.8 Case Study

Fig. 5.9: Visualization of the clusters found with k-means with \( k = 6 \) (colors) and the labels (markers). We observe 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, standing, and washing hands seem to occur equally frequent among clusters four and six. We can instantly see 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 Figure 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, again \( k = 6 \) is best, see Figure 5.11.

The best silhouette score obtained is 0.742, the same as 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. Also from the table we do not observe any significant differences.
k-means clustering (5)

- CrowdSignals accelerometer data

![Silhouette plot for various clusters.](image)

Table 5.1: Distribution of measurements and labels over clusters. Note that the percentage for the label indicates the percentage of total rows among which the label has been assigned.
Table 5.1: Distribution of measurements and labels over clusters. Note that the percentage for the label indicates the percentage of total rows among which the labels has been assigned.

<table>
<thead>
<tr>
<th>attribute</th>
<th>statistic</th>
<th>cluster 1</th>
<th>cluster 2</th>
<th>cluster 3</th>
<th>cluster 4</th>
<th>cluster 5</th>
<th>cluster 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Accelerometer data</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acc_phone_x</td>
<td>mean</td>
<td>-0.36</td>
<td>8.24</td>
<td>8.00</td>
<td>-0.75</td>
<td>-8.21</td>
<td>-0.56</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>1.15</td>
<td>0.96</td>
<td>1.02</td>
<td>1.71</td>
<td>0.94</td>
<td>1.21</td>
</tr>
<tr>
<td>acc_phone_y</td>
<td>mean</td>
<td>0.98</td>
<td>0.61</td>
<td>-2.35</td>
<td>-9.61</td>
<td>2.25</td>
<td>9.55</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>1.92</td>
<td>1.36</td>
<td>2.06</td>
<td>1.22</td>
<td>1.94</td>
<td>1.12</td>
</tr>
<tr>
<td>acc_phone_z</td>
<td>mean</td>
<td>9.19</td>
<td>4.54</td>
<td>-4.80</td>
<td>0.23</td>
<td>-4.67</td>
<td>-0.56</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>1.06</td>
<td>1.36</td>
<td>1.08</td>
<td>1.49</td>
<td>1.44</td>
<td>1.62</td>
</tr>
<tr>
<td><strong>Labels</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>labelOnTable</td>
<td>percentage</td>
<td>99.56 %</td>
<td>0.44 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
</tr>
<tr>
<td>labelSitting</td>
<td>percentage</td>
<td>2.40 %</td>
<td>0.40 %</td>
<td>97.20 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
</tr>
<tr>
<td>labelWashingHands</td>
<td>percentage</td>
<td>7.02 %</td>
<td>1.75 %</td>
<td>1.75 %</td>
<td>56.14 %</td>
<td>0.00 %</td>
<td>33.33 %</td>
</tr>
<tr>
<td>labelWalking</td>
<td>percentage</td>
<td>1.87 %</td>
<td>0.94 %</td>
<td>0.47 %</td>
<td>46.14 %</td>
<td>0.47 %</td>
<td>50.12 %</td>
</tr>
<tr>
<td>labelStanding</td>
<td>percentage</td>
<td>4.74 %</td>
<td>1.42 %</td>
<td>0.00 %</td>
<td>48.34 %</td>
<td>0.47 %</td>
<td>45.02 %</td>
</tr>
<tr>
<td>labelDriving</td>
<td>percentage</td>
<td>1.67 %</td>
<td>55.56 %</td>
<td>20.00 %</td>
<td>0.00 %</td>
<td>22.22 %</td>
<td>0.56 %</td>
</tr>
<tr>
<td>labelEating</td>
<td>percentage</td>
<td>2.54 %</td>
<td>39.59 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>57.36 %</td>
<td>0.51 %</td>
</tr>
<tr>
<td>labelRunning</td>
<td>percentage</td>
<td>17.43 %</td>
<td>0.92 %</td>
<td>17.43 %</td>
<td>64.22 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
</tr>
</tbody>
</table>

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, again \( k = 6 \) is best, see Figure 5.11. The best silhouette score obtained is 0.742, the same as 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. Also from the table we do not observe any significant differences.

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 silhouette score always 0.679. Based on the silhouette, we decide to use the k-means clustering approach to add a feature which represents the attribution of an instance in the dataset to a cluster.
k-medoids clustering (1)

• k-medoids: same as k-means, except that we do not use artificial mean centers, but actual points

• k-means: suitable for person level distance metrics?
  – k-medoids is known to be more suitable
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 = centers
while prev_centers != centers do
    prev_centers = centers
    cluster_assignment = []
    for $i = 1, \ldots, N$ do
        cluster_assignment[$i$] = argmin$_{j=1,\ldots,k}$ distance($x_i$, centers[$j$])
    end
    for $j = 1, \ldots, k$ do
        cluster_center[$j$] = 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
Hierarchical clustering (1)

- Can also take a more iterative approach:
  - Divisive clustering
    - Start with one cluster, make a split in each step
  - Agglomerative clustering
    - Start with one cluster per instance and merge
  - Can be visualized using a dendrogram
Hierarchical clustering (2)

Algorithm 5: k-medoids clustering

for \( i = 1, \ldots, k \) do

1. \( \text{centers}[k] = \) random point from \( x_1, \ldots, x_N \) not part of centers yet

end

\( \text{prev centers} = \text{centers} \)

while \( \text{prev centers} \neq \text{centers} \) do

1. \( \text{prev centers} = \text{centers} \)

2. \( \text{cluster assignment} = [] \)

3. for \( i = 1, \ldots, N \) do

   a. \( \text{cluster assignment}[i] = \arg\min_j \{ \text{distance}(x_i, \text{centers}[j]) \} \)

4. for \( j = 1, \ldots, k \) do

   a. \( \text{cluster center}[j] = \arg\min_l \{ \sum_{i | \text{cluster assignment}[i] = j} \text{distance}(x_l, x_i) \} \)

end

end

going from one cluster at the top to the most refined clusters at the bottom. This allows one to select the level of clustering which is appropriate for the domain. For a more extensive overview of the approaches, see [63].
We need to know which clusters to merge

### Different criteria:

- **Single linkage**
  \[ d_{SL}(C_k, C_l) = \min_{x_i \in C_k, x_j \in C_l} \text{distance}(x_i, x_j) \]

- **Complete linkage**
  \[ d_{CL}(C_k, C_l) = \max_{x_i \in C_k, x_j \in C_l} \text{distance}(x_i, x_j) \]

- **Group average**
  \[ d_{GA}(C_k, C_l) = \frac{\sum_{x_i \in C_k} \sum_{x_j \in C_l} \text{distance}(x_i, x_j)}{|C_k| \cdot |C_l|} \]
Agglomerative clustering (2)

• We need to know which clusters to merge
• Different criteria:
  – Ward’s criterion (minimize standard deviation):

\[
d_{\text{Ward}}(C_k, C_l) = \sum_{x_i \in C_k \cup C_l} ||x_i - m_{C_k \cup C_l}||^2 - \sum_{x_j \in C_k} ||x_j - m_{C_k}||^2 - \sum_{x_n \in C_l} ||x_n - m_{C_l}||^2
\]
Agglomerative clustering (3)

Algorithm 6: Agglomerative clustering

\[
\text{clusters} = \{\} \\
\text{for } x_i \in X \text{ do} \\
\quad \text{clusters} = \text{clusters} + \{x_i\} \\
\text{end} \\
\text{while True do} \\
\quad C_k, C_l = \arg\min_{C_k \in \text{clusters}, C_l \neq C_k \in \text{clusters}} d(C_k, C_l) \\
\quad \text{if } d(C_k, C_l) > th \text{ then} \\
\quad\quad \text{return clusters} \\
\quad \text{else} \\
\quad\quad \text{clusters} = \text{clusters} \setminus \{C_k\} \\
\quad\quad \text{clusters} = \text{clusters} \setminus \{C_l\} \\
\quad\quad \text{clusters} = \text{clusters} + \{C_k + C_l\} \\
\quad \text{end} \\
\text{end}
\]
Agglomerative clustering (4)

- CrowdSignal example (Ward’s criterion)

Fig. 5.12: Dendrogram for the crowdsignals dataset

5.9 Exercises

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 and that would be the distance between the attribute value of two persons. From the literature, it is known that shortest path is not always be the best option to use as a distance metric. Give an example that supports this statement. Furthermore, give an alternative dynamic time warping distance metric that takes away 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?
Divisive clustering (1)

- We start with a big cluster and split
- We define a dissimilarity of a point to other points in the cluster C:
  \[ \text{dissimilarity}(x_i, C) = \frac{\sum_{x_j \neq x_i \in C} \text{distance}(x_i, x_j)}{|C|} \]
- We create a new cluster C’ and remove the most dissimilar points from C to C’ until these are more dissimilar to points in C’
- What C should we select?
  - One with the largest diameter:
  \[ \text{diameter}(C) = \max_{x_i, x_j \in C} \text{distance}(x_i, x_j) \]
Divisive clustering (2)

Algorithm 7: Divisive clustering

clusters = \{ \{x_1, \ldots, x_N\} \}

while |clusters| < N do
  \( C = \arg\max_{C \in \text{clusters}} \text{diameter}(C) \)
  clusters = clusters \ C
  most_dissimilar_point = \arg\max_{x \in C} \text{dissimilarity}(x, C)
  \( C = C \cap \text{most_dissimilar_point} \)
  \( C_{\text{new}} = \{\text{most_dissimilar_point}\} \)
  point_improvement = \infty
  while point_improvement > 0 do
    \( x = \arg\max_{x \in C} \text{dissimilarity}(x, C) \)
    \( \text{distance}_C = \text{dissimilarity}(x, C) \)
    \( \text{distance}_{C_{\text{new}}} = \text{dissimilarity}(x, C_{\text{new}}) \)
    point_improvement = \text{distance}_{C_{\text{new}}} - \text{distance}_C
    if point_improvement > 0 then
      \( C = C \setminus x \)
      \( C_{\text{new}} = C_{\text{new}} + x \)
    end
  end
  clusters = clusters + C + C_{\text{new}}
end

return clusters

In other words, the diameter is the maximum distance between points in the cluster. Formally, the algorithm is then defined as follows:

\[ \text{diameter}(C) = \max_{x, j} d(x_i, x_j) \quad (5.22) \]

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 our clustering. Users could do this, but the task is not clearcut. Other dimensionality reduction approaches such as Principal Component Analysis can also be used, but do not provide intuitive results either as we are transforming the attributes that initially had meaning to a less meaningful new space. The domain of subspace clustering comes to the rescue here. We will
Subspace clustering (1)

• What if we have a huge number of features?
• Would everything even out?
  – Very likely
  – Most clustering approaches do no handle a large number of features well

• Subspace clustering can provide a solution:
  – Looks at subspace in the feature space
  – Focus on the CLIQUE algorithm (cf. Agrawal, 2005)
Subspace clustering (2)

• First step: we create units based on our features (ε distinct intervals per feature)
Subspace clustering (3)

• A unit is defined by upper and lower boundaries per feature: $u = \{u_1, \ldots, u_p\}$
• $u_i(l)$ is the lower bound, $u_i(h)$ the upper bound for the value of attribute $i$
• An instance $x$ is part of the unit when its value fall in the boundaries:

$$\text{belongs to}(x, u) = \begin{cases} 1 & \forall i \in 1, \ldots, p u_i(l) \leq x^i < u_i(h) \\ 0 & \text{otherwise} \end{cases}$$
Subspace clustering (4)

- We can define the selectivity of a unit as:

\[
selectivity(u) = \frac{\sum_{i=1}^{N} \text{belongs to}(x_i, u)}{N}
\]

- And given a threshold \( \tau \) a unit is dense when:

\[
dense(u) = \begin{cases} 
1 & selectivity(u) > \tau \\
0 & \text{otherwise}
\end{cases}
\]

- Previous figure showed dense units (grey)
Subspace clustering (5)

- We want subspaces (subsets of attributes), so our units do not have to cover all attributes.
- We can have units that cover \( k \) attributes.
- Units have a common face when:

\[
\text{common\_face}(u_1, u_2) = \begin{cases} 
1 & \exists i \in 1, \ldots, k : (r_i(l) = r_i'(h) \lor r_i(h) = r_i'(l)) \land \\
& \forall j \neq i \in 1, \ldots, k : (r_i = r_i') \\
0 & \text{otherwise}
\end{cases}
\]

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

\[
\text{connected}(u_1, u_2) = \begin{cases} 
1 & \exists i \in 1, \ldots, k : (r_i(l) = r_i'(h) \lor r_i(h) = r_i'(l)) \land \\
& \forall j \neq i \in 1, \ldots, k : (r_i = r_i') \\
0 & \text{otherwise}
\end{cases}
\]

If we consider dense units as defined previously that are connected in our figure we obtain a cluster in the upper right and one in the lower left. 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

\[
(0.6 \leq X_1 < 1) \land (0.6 \leq X_2 < 1)
\]

but also by

\[
(0.6 \leq X_1 < 0.8) \land (0.6 \leq X_2 < 1)
\]

\[
(0.6 \leq X_1 < 1) \land (0.6 \leq X_2 < 1)
\]

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? That will be the first problem we tackle, to find these in an efficient way. 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 draw on the basis 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.

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 \( R_k \) the set of dense units of dimensions \( k+1 \). \( u_i(a) \) refers to the name of the \( i \)th attribute of unit \( u \).

We see that 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 \( k+1 \). An ordering is...
Subspace clustering (6)

• Units are connected in a cluster when they share a common face or when they share a unit that is a common face to both:

\[
\text{connected}(u_1, u_2) = \begin{cases} 
1 & \text{common\_face}(u_1, u_2) \lor \\
\exists u_3 : \text{common\_face}(u_1, u_3) \land \text{common\_face}(u_2, u_3) & \\
0 & \text{otherwise}
\end{cases}
\]

• We strive to find dense units using a combination of attributes to form the clusters
Subspace clustering (7)

• We start with one attribute and work our way up using some fancy algorithms
• Once we have found all the dense units we find the clusters
• The details on the precise computational steps are provided in the book