PV251 Visualization

Autumn 2024

Study material

Lecture 3: Data preprocessing

In this lecture, we will present different approaches to data preprocessing that prepare the input dataset for the subsequent visualization (e.g., handles the erroneous or missing data).

It is generally preferred to display original data without any modification. The main reason is fear of losing important information or adding undesirable artifacts. An example is medical visualization. Viewing rough data often identifies problem locations in data logs, such as missing data or significantly different records that may indicate bad calculations.
On the other hand, some of the preprocessing methods are necessary for other types of data.

Data preprocessing techniques:

1. Metadata and statistics
2. Missing values and data „cleaning“
3. Normalization
4. Segmentation
5. Sampling and interpolation
6. Dimension reduction
7. Data aggregation
8. Smoothing and filtration
9. Conversion of raster data to vectors

1. Metadata and statistics

Information about data sets – metadata and statistical analysis can provide invaluable information for pre-processing of data. Metadata provides information that can help in interpreting data (for example, the format of individual records). In addition, they may contain a reference point from which the data field records are measured, the unit of such measurement, the symbol or number used to indicate the missing value and the resolution at which the data was acquired. Such information may be important for selecting the appropriate preprocessing method and setting its parameters.
Different methods of statistical analysis can provide a useful data preview.

- *Outlier detection* can identify data with wrong data fields.

- *Cluster analysis* can help in data segmentation into groups according to their similarity.
- *Correlation analysis* enables to remove the redundant fields or to highlight the association between dimensions, which does not have to be clear on the first sight.

2. Missing values and data „cleaning“

A frequent phenomenon in the analysis and visualization of real data sets is the absence of some data records or their erroneous value. The reason for the absence of data can be, for example, an error in the data sensor or a blank check box in the questionnaire. The reason for erroneous data is most often a human factor and is difficult to detect. In both cases, however, when analyzing such data, it is necessary to choose a strategy to deal with these errors. Some of these strategies, especially those that link to visualization, will now be addressed.

1) **Removing wrong records.** This seemingly drastic method of removing all missing or bad fields is in fact one of the most common. Of course, the quality of the information displayed is questionable. It can lead to a significant loss of information because it is reported that in some sectors up to 90% of records contain at least one wrong field. Missing data records may be some of the most interesting ones.
2) **Assigning a defined value.** Each variable in the dataset determines the constant value we assign to it if we find the disputed value in the record. For example, for a variable with a range of values between 0 and 100, we can choose a missing value -5, for example. Then, when visualizing such data at a glance, we see which data was problematic. It is obvious that for the subsequent statistical analysis it is necessary to omit these artificially added values.
3) **Assigning an average value**. A simple strategy to deal with missing data is to replace them with the average value of a given variable or dimension. The advantage of this approach is to minimize the overall statistics calculated for the given variable. The disadvantage of this approach is that the selected value may not always be "correct". Another drawback is the fact that we get rid of potentially interesting places in the data.
4) **Assigning a value derived from the nearest neighbor**. Better approximation is to find the most similar record. Similarity is based on the analysis of differences in all other variables. The principle is as follows. Consider the entry A, which lacks the input for variable *i*. Also, take a record B that is closer to A than all other records in the dataset, taking into account all other variables except for *i*. In this case, in the variable even in the record A, we put the value of the variable from the record B. The problem with this solution is the fact that the variable *i* can only depend on a small subset of the other record variable (s), so the "best neighbor" strategy may not always find the best solution.

5) **Calculating of imputation**. A very complex process, with long-term research behind it. The process of calculating the refund of missing or missing data is known as imputation = substitution.

3. Normalization

Normalization is the process of transforming an input data set in such a way that the transformed data matches predetermined statistical properties. A typical simple example of transformation is the conversion of the range of input data values only to the interval [0.0, 1.0]. Other forms of normalization convert data in such a way that each variable (dimension) has a mean and standard deviation. Normalization is a very useful operation because it makes it possible to compare seemingly incomparable variables. It is also very important in the visualization process when individual graphical attributes only acquire certain possible values, so it is necessary to map the input data to these attributes. This leads to the conversion of the data range to the range of graphical attributes.
For example, if we take d_{min} and d_{max} the minimum and maximum values in the variables, we can normalize all values in the range from 0.0 to 1.0 using the formula:

$$d_{normalized} = (d_{original} - d_{min})/(d_{max} - d_{min})$$

In some cases, we set the scale and offset to simplify data interpretation in such a way as to match intuitive minimum and maximum values. An example may be a dataset whose values fall within a percentage range of 40 and 90. In this case, it is more intuitive to scale the data to 0-100%.
Normalization can also include trimming at **threshold values** where threshold values above that threshold are closed to this threshold (they are assigned a threshold value). This is advantageous, for example, in mapping such data to graphical attributes in visualization.

4. Segmentation

In many cases, data may be divided into continuous areas where each area corresponds to a certain classification of data. For example, magnetic resonance data originally has 256 values that a given data point can take and then such a point can be segmented into one of the specified categories, such as bone, muscle, fat, skin, etc. Simple segmentation can be done by simply mapping discontinuous ranges of data values into these given categories.
However, in most cases, such mapping is ambiguous. In such cases, it is necessary to look at the classification of adjacent points, which will improve the quality of the entire classification. Another possibility is **probabilistic segmentation** where each point is assigned a probability that belongs to that category.
A typical segmentation problem is so-called **subsegmentation** (large areas of unconnected regions) or **over-segmentation** (a large number of small regions that are related). The solution is iterative repetition of the split-and-merge segmentation process.

Split-and-merge

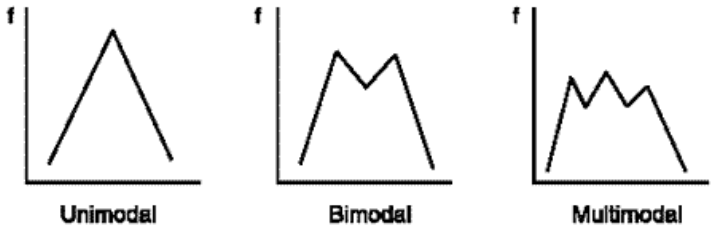
- similarThresh = defines the similarity of two regions with given characteristics
- homogeneousThresh = defines the region homogeneity (uniformity)

```
do {
      changeCount = 0;
      for each region {
            compare region with neighboring ones and find the most
similar one;
            if the most similar one is within similarThresh of the
current region {
                  connect these two regions;
                  changeCount++;
            }
            evaluate the homogeneity of the region;
            if homogeneity of region is smaller than
homogeneousThresh {
                  split the region to two parts;
                  changeCount++;
      }
} until changeCount == 0
```

The iterative split-and-merge algorithm works as follows. At the beginning, thresholds *similarThresh* are defined, defining the similarity of two regions and *homogeneousThresh*, which defines homogeneity within a region. Then, the algorithm itself runs through each region in the given input dataset. Such a region compares with neighboring regions and, according to a predefined similarity, finds the most similar. If this neighbor is found within a tolerance defined by a *similarThresh* threshold, these two regions are joined together. In this way, regions are merged. The second option that can occur is the distribution of a region that is not homogenous enough (defined by the *homogeneousThresh* threshold). At each change (casting or distribution), the change is recorded in the algorithm using the *changeCount* variable. If no change occurs in the new iteration, the value of this variable is not changed and the algorithm ends.

The most difficult parts of the algorithm are:

1) Determining the similarity of two regions. The easiest method is to compare the average values in each region.
2) Evaluation of region homogeneity. The solution can be to evaluate the histogram of values within this region and determine whether it is unimodal or multimodal.



3) Division of the region. A typical algorithm creates two (or more) subregions at points (points) where the most different values are found. These points are then marked as *seeds*, and the filling process is started when the regions "grow" until all points of the input set are allocated somewhere.

The algorithm may be prone to the problem of creating an infinite loop where we constantly divide and connect the same region. A simple solution is to change the threshold of similarity or homogeneity. More sophisticated algorithms include other region properties such as smoothness of boundaries or region size and shape.

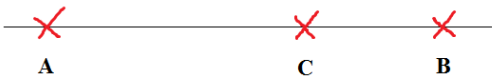
5. Sampling and interpolation

Often, it is necessary to transform a data set defined with a certain spatial layout into another data set with a different resolution. For example, let's give an image that we want to reduce or enlarge. Another example is an image where only a few input point samples are known and we want to add values for the points placed between these samples. In both cases, we assume that the data we have at the input represent discrete samples of continuous space, so we can predict values at other points by examining the closest specified points. The addition of such data is referred to as **interpolation**. This is a commonly used method that is used in many areas, including visualization. The most common interpolation techniques are as follows:

- linear
- bilinear
- non-linear

Linear interpolation

Suppose we have the value of the variable d in two places, A and B. Using linear interpolation, we estimate the value of this variable at the C point between the A and B points.



First, we calculate the percentage C between A and B. This is then used in conjunction with the magnitude of the change in the value of the variable between A and B, and the value d in C. Assuming that all three points lie in the x-axis, the following equation applies:

$$(x_C - x_A)/(x_B - x_A) = (d_C - d_A)/(d_B - d_A) \text{ or}$$

$$d_C = d_A + (d_B - d_A)*(x_C - x_A)/(x_B - x_A)$$

This is similar to the normalization we discussed earlier. To remove dependence on the x-axis, we can use parametric equations that define the change of position and the change of value between A and B. From them we calculate the value of the parameter in the equation that defines the point C and this number is used to calculate the value in point C. Parametric form of equations is the following:

$$P(t) = P_A + Vt, \text{ where } V = P_B - P_A$$

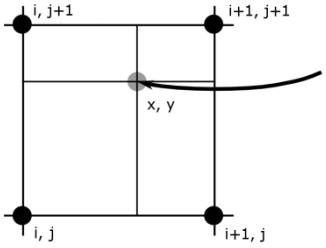
Note that we have never defined the number of dimensions of the space in which we are working. This is because these calculations work in the space of any dimension.

If we put the PC on the left side of the equation, we can calculate the value t and then use it in calculating the value change:

$$d(t) = d_A + Ut, \text{ where } U = d_B - d_A.$$

Bilinear interpolation

We can extend the previous concept to two or more dimensions by repeating the procedure for each dimension. For example, a common task in 2D space is to calculate the value of d in the position (x, y) in the uniform grid (point distribution is uniform in both axes). This is the case for 2D images. If the position (x, y) for which we look for a value corresponds to a point in the grid, then the value sought is exactly the value of the given grid point. However, if the search position lies between the points in the grid, the value must be calculated. To do this, it is necessary to find four adjacent grid points that surround the point (x, y). Assuming that the lattice point positions are represented by integer values and that the distance between the points is 1.0 and the search value (x, y) has both fragments, then the bounding box formed by the grid points containing the point sought (x, y) (i, j + 1) and (i + 1, j + 1) where i is the largest integer smaller than x and j is the largest integer smaller than y.



Now we will interpolate in this space. First we perform horizontal interpolation, then vertical. Using the linear interpolation described above, we calculate the percentage point x between points i and i + 1. We can denote this value as s. Now we can calculate the value d in positions (x, j) and (x, j + 1) using values at four border points. Similarly, we calculate the percentage of the y point between points j and j + 1, we denote it as t. Finally, we calculate the value in position (x, y) by interpolating the calculated values obtained from the horizontal interpolation and the value t, so:

$$d_{x,y} = d_{x,j} + t * (d_{x,j+1} - d_{x,j})$$

dx,y is the weighted average of the four boundary values with respect to the position of the point (x, y).

Non-linear interpolation

One of the main problems of linear interpolation is the fact that while local value changes have a smooth transition, changes on the opposite side of the grid point can be significantly different. In fact, the continuity at the grid point is 0. This can be improved by using a different type of interpolation – using higher order polynomials. This includes quadratic and cubic curves – splines. Their main purpose is smooth interpolation in points, giving the

control points of the curve and the blending function. You can see an example of a smooth connection of the quadratic and cubic Bezier curves.

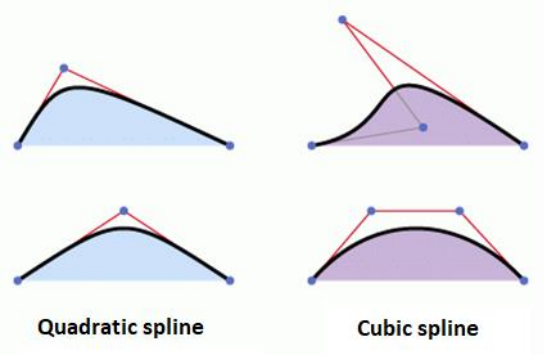
To remind:

Continuity C0 = the endpoint of the first segment is the starting point of the second segment.

Continuity C1 = the tangent vector at the endpoint of the segment is equal to the tangent vector at its starting point.

Continuity C2 = Equation of the vector of the first and second derivatives is required.

etc.

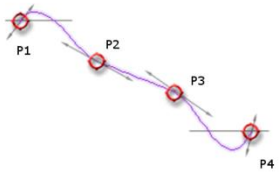


Catmull-Rom spline

Let's have 4 control points (p_0, p_1, p_2, p_3). Then the cubic Catmull-Rom spline passing through these point can be defined as:

$q(t) = 0.5 * ((2*p_1) + (-p_0 + p_2) * t + (2*p_0 - 5*p_1 + 4*p_2 - p_3) * t^2 + (-p_0 + 3*p_1 - 3*p_2 + p_3) * t^3)$

$$q(t) = 0.5 * (1.0\ t\ t^2\ t^3) * \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} * \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$



The result of the interpolation application using Catmull-Rom curves can be shown in the following example. The task is to magnify the original 24x24 pixels. The result is a comparison of the results using a cubic B-spline filter and Catmull-Rom.

Original image (24x24 pixels)

cubic B-spline filter

Catmull-Rom

Resampling methods

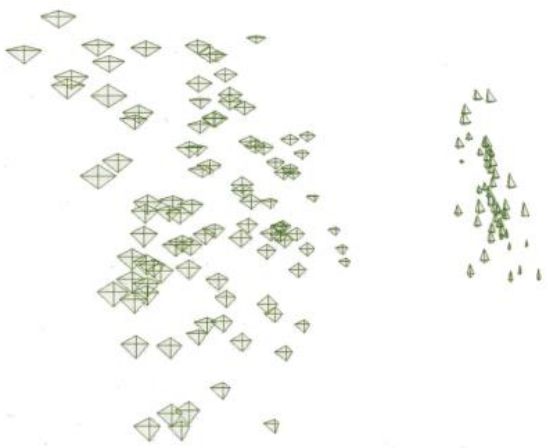
Depending on the input data density, two operations are possible: reducing the size of the input data (for example, reducing the input image) or replicating the data to increase the input data set (as in the case shown on the slide). **Subsampling** – the first case – can have a very simple solution. For example, we select only every n-th record from the input dataset. However, it is clear that some important aspects of input data may be lost. As an example, let us list every fourth point on the map. We do not guarantee that important information (e.g., path) is not contained in the three points removed. Better approaches include neighboring averaging, median selection, or random data selection in a subregion defined around the surveyed value.

Another frequently used resampling method is "**data subsetting**". This method is mainly used for processing large datasets, where visualization of such a set can lead to unpleasant visual artefacts (e.g., when overloading memory, etc.). The user can specify the set of queries that will be used to filter the input set. An example may be a filter that returns data taken only within a certain time period. Thus, the threshold is set to a particular attribute. Subsetting can be used not only during the preprocessing phase but also during the visualization itself, when the user interacts with displayed data (highlighting, drawing, selection, etc.). Interactive subsetting is generally more effective than query-based subsetting because the user can directly control and decide on input data filtering. On the other hand, the advantage of the query-based approach is that we do not have to pull the entire input set into the program.

6. Dimension reduction

If the dimensionality of the input data exceeds the capabilities or capabilities of the visualization technique, it is necessary to find a way to reduce this dimensionality. Of course, it is necessary to keep as much information as possible in the input set. Approaches to solving this problem can be divided into two groups:

1) Manual approach where the user has to enter those dimensions that are indispensable for him in the given situation.
2) Automatic calculation techniques such as **PCA** (principal component analysis), **MDS** (multidimensional scaling) and **SOMs** (Kohonen self-organizing maps). All of these methods are capable of preserving most of the essential properties of an input set, such as clusters, patterns, or contours. However, the output of these methods is strongly dependent on input configuration and calculation parameters, resulting in other results.



The figure shows the result of reducing the four-dimensional data set to 2D space using the PCA method. The individual records are shown here using a pictogram representing 4 variables (one for each dimension). The size of the value in the variables is given by the size of the line with the beginning at the center of the pictogram and the end at the appropriate vertex.

PCA (Principal Component Analysis)

PCA creates additional attributes that are a linear combination of the original data variables. These new attributes define a subspace of variables that minimizes the average error of the lost information. PCA has the following steps:

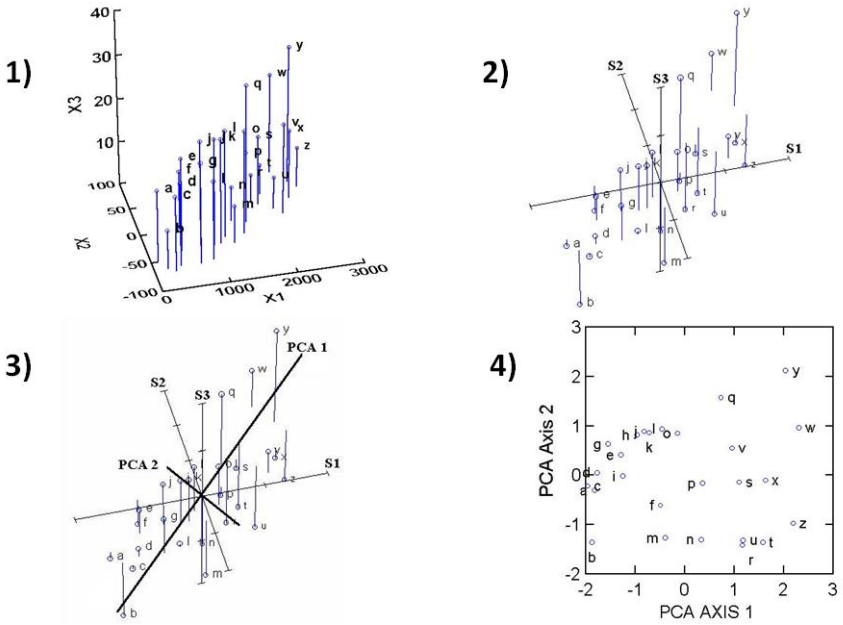
1) Assume the input data has m dimensions / attributes. From each item of each record, the average of the dimension is subtracted. The result is a dataset with a mean value equal to zero.
2) Calculate the covariance matrix (a square matrix describing the dependence of a set of random variables).
3) We calculate the eigenvectors and the eigenvalues of this covariance matrix.
4) We sort our eigenvectors based on their eigenvalues - from the largest to the smallest.
5) We select the first mr of custom vectors, where mr is the number of dimensions on which we want to reduce the input data.
6) Create an array of these eigenvectors, where the vectors form the matrix rows and the first eigenvector represents the first line of this matrix.
7) For each data record we create a vector of its values, transpose it and multiply with the previous matrix. This will result in the resulting transformation of the record – each record is now converted into a reduced space.

PCA more intuitively:

1) We select a line in space visualizing n-dimensional data. This line covers the most of the input data items and is called the first principal component (PC).
2) We select a second line perpendicular to the first PC, this forms the second PC.

3) We repeat this until we proces all PC dimensions or until we reach a desired number of principle components.

Here's an example of PCA usage. Figure 1 shows a hypothetical example of the result of a measurement of three different animal species X1, X2 and X3. In this example, it is difficult to see that types X1 and X2 are in some relationship. However, it is almost impossible to evaluate any relationship between X3 and the remaining two species from this representation.
In the first phase (Figure 2), we rotate the data set by subtracting the mean deviation and dividing the standard deviation. This is called standardization. Thus, the center of gravity of the entire dataset is set to zero. Standardized axes are designated as S1, S2 and S3. The relative position of individual data remains the same. From this representation, it is already possible to observe a gradient going from the lower front corner to the upper rear. It can be seen that along this gradient the values for species X1 and X2 increase.



The PCA method selects the first PCA line (axis) as the line passing through the center of gravity, which at the same time minimizes the square of the distance of each of the points to that line. In other words, this line is closest to all of the data as it goes.
The second PCA axis must also pass through the center of gravity, and must meet the same conditions, but only the limitation of its position relative to the first PCA axis (for example, it must be perpendicular).
If we now rotate the coordinate system defined by PCA1 and PCA2, when we define that PCA1 corresponds to the x axis and the PCA2 axis y, we obtain the diagram in Figure 4.

MDS (Multidimensional Scaling)

Another method for reducing the input data dimension is multidimensional scaling (MDS), also known as the *gradient descent approach*. MDS tries to find a representation of input

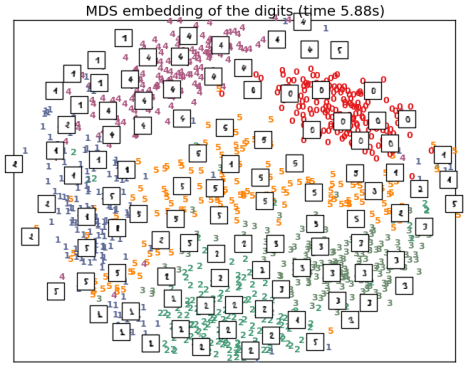
data in a lower dimension that best preserves the distances between the individual input data points of the original set. In other words, the goal for each pair of points (i, j) is to reduce the difference between $d_{i,j}$ (the distance between i and j in the original n-dimensional space) and $delta_{i,j}$ (the distance between these points in the reduced space). This distance difference is referred to as *stress*.

The algorithm works as follows:

1) We calculate the distances between all pairs of data points in the original space. If we have *n* points as an input, this step requires n(n – 1)/2 operations.
2) We transfer all input data points to points in the reduced dimension space (often randomly).
3) We calculate *stress*, i.e., difference in distance between points in the original and reduced space. This can be done using different approaches.
4) If the average and cummulated *stress* value is smaller than the user-defined threshold, the algorithm ends and returns the result…
5) If the *stress* value is higher than the threshold, for each point we calculate a directional vector pointing to the desired shift direction in order to reduce *stress* between this point and the other points. This is determined as the weighted average of vectors between this point and its neighbors and its weight is derived from *stress* value calculated between individual pairs. Positive *stress* value repulses the points, negative one attracts them. The higher the absolute value of *stress,* the bigger movement of point.
6) Based on these calculations we transform tha data points to the target reduced dimension, according to the calculated vectors. Return to step 3 of the algorithm.

Potential problems of this algorithm are the creation of endless loop or getting stuck in a non-local optimum. This can be avoided by repeating the algorithm with different input parameters or allowing "non-standard" movement in a different direction than the calculated directional vector.

Many visualization and statistical graphical packages use both methods – PCA and MDS. Some even use a combination of both, when the PCA is first used to calculate the initial point positions to which the MDS method is then applied. This leads to a large reduction in the number of iterations necessary to achieve the lowest *stress* configuration.



A practical example of using the MDS algorithm can be to reorganize a photo album by similarity – in this case, similarity is defined by color. The image is the output of Yorg (http://lear.inrialpes.fr/src/yorg/doc/index.html).

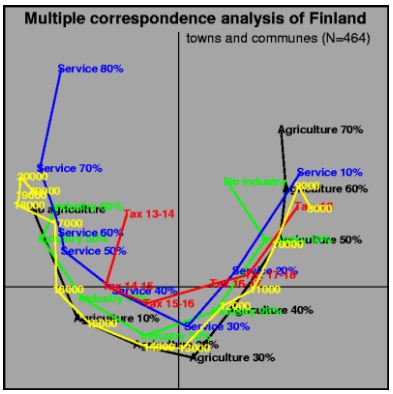
Conversion of raster data to vectors

There are many areas where one or more dimensions of a dataset contains nominal values. To process such data, it can be chosen from several strategies, depending on the characteristics of these data - the number of such dimensions, how wide the values can take values, or whether the values can be sorted or defined. The key is to find a way of mapping data to graphical entities or attributes that do not relate to data that are not present in the original data.

Take an example of a car database. The data set containing the information about the individual cars contains the "manufacturer" and "car model" nominal values. How should we map these values to a chart? A possible way is to assign an integer value for each of the nominal values - alphabetically, for example. But this may lead to mistaken relationships, for example Honda will be closer to Ford than to Toyota in the chart. There is no such relationship in fact.

If there is one nominal variable in the input dataset, there are several possible techniques that we can use. The simplest is to use this variable as the label of the graphic element we display. This method is suitable for small datasets, as it grows quickly becomes unusable. Modern techniques are able to locate this method, which means, for example, displaying a label only for variables near the cursor and the like.

When searching for a method for mapping a nominal variable to a number, we can control the similarity between the numerical variables associated with these nominal ones in a single data record. Then it is clear that records with similar properties should be the nominal values of these records mapped to a similar numerical value. If we have a similarity between all pairs of data records, we can use MDS, for example, and map nominal values to positions in one dimension. This is a simplified technique called **corespondence analysis** that is used in statistics. This technique can also be used in cases where all dimensions of the input data set are nominal. Then this technique is called **multiple correspondence analysis**.

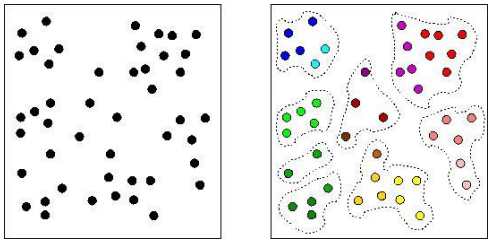


7. Data aggregation

When processing large sets of input data, it is very useful to aggregate these data into clusters according to the similarity of their values and / or position. These groups are then represented by a much smaller amount of data. This can be achieved by a simple averaging of values, but new records may include more descriptive data such as the number of members of each group or the range of their positions or values. This is called **aggregation** (clustering).

Therefore, the aggregation method has two components: the point aggregation method and the display method of the resulting groups. Clustering can be done in a variety of ways, such as neighboring points, space dividing, or iterative split-and-merge methods. An important aspect of all methods is the calculation of the distance between data points and the quality of clustering, including the definition of separation of individual clusters.

When visualizing clusters, it is necessary to provide the user with enough information to decide whether or not they need to examine the particular clusters. Just displaying one representative of a given cluster is usually not enough to understand the variability of data inside the cluster.



8. Smoothing and filtration

Signal processing is a common process of smoothing input data. The goal is to reduce noise and blur sharp discontinuities. Typically, smoothing is performed using a so-called convolution. For our purposes, it is sufficient if we look at the convolution as the weighted average of the neighbors surrounding the given point. In a one-dimensional space, the convolution can be applied using the following formula:

$$p_i = \frac{p_{i-1}}{4} + \frac{p_i}{2} + \frac{p_{i+1}}{4}$$

where each p_i corresponds to a processed point.
After applying this operation to a given point, its original value, which differs significantly from its neighbors, is replaced by a value much more similar to that of the neighbor. Changing the scales or the shape and size of the surrounding area can change the resulting image.

9. Conversion of raster data into vector ones

In computer graphics, objects are typically represented by a set of polygons formed by vertices and edges. The task is to create a raster representation of such objects at the pixel level, define their surface properties, define their interaction with light and other objects. In certain cases, it is advantageous to extract linear structures from a raster data set (e.g., from the figure). The reasons for this conversion can be as follows:

- Content compression, e.g., for transmission. The list of vertices and edges is almost always a more compact expression than a raster representation.
- Compare the content of two or more images. It is easier to compare higher order attributes than individual pixels.
- Data transformation. Afin transformation, such as rotation and scaling, is easier to apply to a vector representation than a raster.
- Segmentation of data. Isolation of regions by highlighting their borders is an effective tool for interactive evaluation and modeling.

Image processing and computer vision are areas where a number of techniques have been developed to convert raster images into their vector form. Let's name some of them:

- **Thresholding**. The principle is the definition of one or more limit values, by which the input set is then divided into several regions. Once borders are determined, individual edges and peaks can be generated. Threshold values can be defined by the user or calculated based on an image histogram analysis. Adaptive thresholding allows you to assign a given threshold to only the area of the image.
- **Region growing**. We start from so-called seeds in the image that are defined by the user or calculated by scanning data. Using seed-filling, we merge pixels into clusters according to their similarity. The main problem is to determine a suitable definition of pixel similarity.

- **Boundary-detection**. This is done by using convolutions of the input image using the corresponding matrix – the convolutional kernel. Each pixel and its neighbors are multiplied by the appropriate value in the convolutional kernel corresponding to their position relative to the processed pixel at the center of the convolutional kernel. These multiples are then added together and the value is assigned to the processed pixel. Various forms of the convolutional kernel allow for various image transformations. For boundary detection, we use a convolution matrix that highlights horizontal, vertical, or diagonal boundary lines and, on the other hand, suppresses pixels whose value is very similar to those in their neighbors.
- **Thinning**. Also used is the convolution process, the goal being to reduce wide linear lines, such as arteries, into lines of one pixel width. These resulting pixels form the central axis of regions that have been thinned.

All the techniques presented in this lecture serve to increase the effectiveness of their visualization and may lead to the discovery of new facts hiding in these data. However, it should be noted that the target user should be informed that the data has been modified in this way. Understanding the different types of transformations that have been applied to the data can significantly help in interpreting them.