RZESZOW UNIVERSITY OF TECHNOLOGY

FACULTY OF MECHANICAL ENGINEERING AND AERONAUTICS

DEPARTMENT OF COMPUTER SCIENCE

DECISION SUPPORT SYSTEMS

LEARNING MATERIALS

PART 1

Time series in machine learning:
time series clustering

Co-funded by
the European Union

COURSE **INDUSTRIAL ENGINEERING**

GROUP **I MP-DU**

Developed by: Łukasz Paśko

2021/2022

Table of contents

Introduction

Objectives of the classes

- Presentation of the concept of time series clustering using machine learning techniques.
- Preprocessing of time series data.
- Application of hierarchical methods.

Time series - a sequence of data (observations) that are **ordered** (the order of the data is important).

Clustering (grouping, patternless classification) - the task of finding clusters, also called clusters or groups, in a set of natural objects.

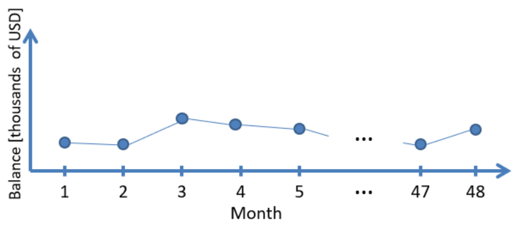
Time series clustering is an important task because:

- Facilitates the discovery of patterns present in the data - generating clusters for a certain set of data makes it easier to understand the structure of the data, allows you to more easily spot anomalies or other types of patterns present in the time series,
- Makes it easier to review large amounts of data - datasets containing time series can be huge (especially nowadays), making it difficult for an analyst to review such a large amount of data,
- Clustering is the most commonly used exploratory technique - it is part of the more complex tasks of:
    - Discovering rules (patterns) in time series,
    - Classification of time series,
    - Detection of anomalies in time series.

The example used in classes

Clustering of bank customers

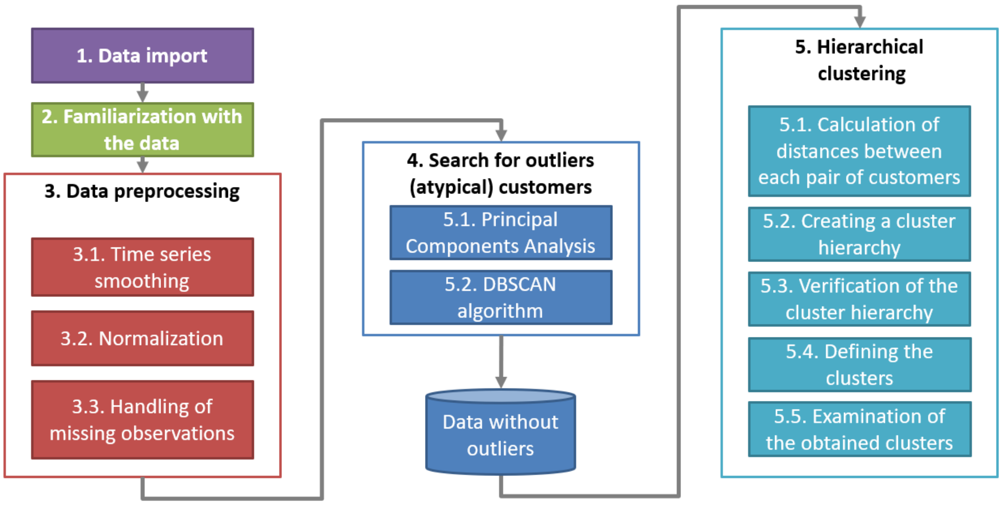
We have data on the values of bank account balances of 5869 customers of a certain bank. These are the values in thousands of USD recorded at the end of the month. The data are for 4 years (48 months).



The account balance data of a single customer is a time series (each customer is one time series). **The task is to find groups made up of similar customers (similar in terms of their account balance in consecutive months), so we are looking for groups of similar time series.**

Customer clustering (dividing customers into groups) is important for various industries and institutions. It is used, among others, in banking, where customer clustering improves, for example marketing campaigns or risk management.

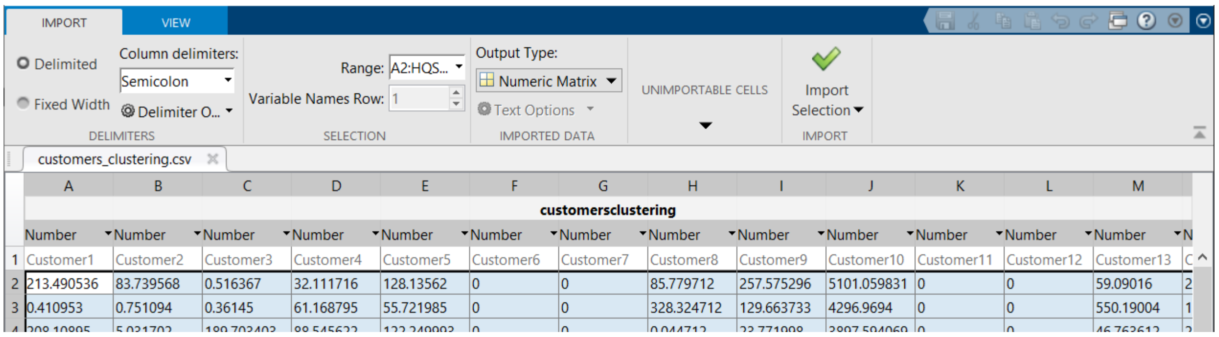
Course of the task


1. Data import
2. Familiarization with the data
3. Data preprocessing
3.1. Time series smoothing
3.2. Normalization
3.3. Handling of missing observations
4. Search for outliers (atypical) customers
5.1. Principal Components Analysis
5.2. DBSCAN algorithm
Data without outliers
5. Hierarchical clustering
5.1. Calculation of distances between each pair of customers
5.2. Creating a cluster hierarchy
5.3. Verification of the cluster hierarchy
5.4. Defining the clusters
5.5. Examination of the obtained clusters

1 Data import

Load the file **customers_clustering.csv** using the import wizard: Home → Import Data.

In the "Range" field enter A2:HQS49 (data range), in the "Output Type" field select "Numeric Matrix". Push the "Import Selection" button, select "Import Data".



Delimited
Fixed Width
Column delimiters: Semicolon
Delimiter O...
Range: A2:HQS...
Variable Names Row: 1
Output Type: Numeric Matrix

customersclustering

	A	B	C	D	E	F	G	H	I	J	K	L	M
	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number
1	Customer1	Customer2	Customer3	Customer4	Customer5	Customer6	Customer7	Customer8	Customer9	Customer10	Customer11	Customer12	Customer13
2	213.490536	83.739568	0.516367	32.111716	128.13562	0	0	85.779712	257.575296	5101.059831	0	0	59.09016
3	0.410953	0.751094	0.36145	61.168795	55.721985	0	0	328.324712	129.663733	4296.9694	0	0	550.19004

2 Familiarization with the data

Check the number of rows and columns of the loaded file.

```
[rows, cols] = size(customersclustering)
```

Draw time series graphs for a few selected customers.

```
hold on
for i=1:3
    plot(customersclustering(:,i))
end
hold off
```

Calculate the average account balance for the first ten customers.

```
mean(customersclustering(:,1:10))
```

Check the maximum and minimum average among all customers.

```
max(mean(customersclustering))
min(mean(customersclustering))
```

Check for missing observations in the data.

```
missing_test = ismissing(customersclustering);
sum(sum(missing_test))
```

Check if there are NaN (not a number) observations in the data.

```
nan_test = isnan(customersclustering);
sum(sum(nan_test))
```

3 Data preprocessing

3.1 Time series smoothing

On the time series of the first client, test the smoothing using the spline functions. Note - you can control the degree of smoothing by adding an option to the following command: `'SmoothingParam', p` where p is the parameter controlling the smoothing. Default value of "p" is 0.9.

```
[f,goodness,output] = fit((1:rows)', customersclustering(:,1),...
    'smoothingspline')  % smoothing of the first series
plot(1:rows, customersclustering(:,1), 'r-', 1:rows, f(1:rows), 'b:')
% comparison of the original and smoothed series
plot(f, 1:rows, customersclustering(:,1), 'residuals' ) % residuals graph
xlabel('Month')
ylabel('Residuals')
```

Smooth out all time series using spline functions.

```
smoothedData=zeros(48,5869);
for i=1:cols  % for all the columns (customers)
    f = fit((1:rows)', customersclustering(:,i), 'smoothingspline');
    smoothedData(:,i) = f(1:rows);
end
```

Check the smoothing effect for the last customer.

```
plot(1:rows, customersclustering(:,cols), 'r-', ...
    1:rows, smoothedData(:,cols), 'b:')
```

3.2 Normalization

Transform the data so that in each time series the maximum value is 1 and the minimum is 0. Formula: $x^* = \frac{x - \min(x)}{\max(x) - \min(x)}$, where x* is the data after normalization, and x is the data before normalization.

```
normalizedData=smoothedData;
for i=1:cols
    normalizedData(:,i)=(smoothedData(:,i)-...
        min(smoothedData(:,i)))/(max(smoothedData(:,i))-...
        min(smoothedData(:,i)));
end
```

Check the effect of normalization for the first three time series, showing them on a single, common graph.

```
plot(normalizedData(:,1),'r-')
hold on
plot(normalizedData(:,2),'g-')
plot(normalizedData(:,3),'b-')
hold off
```

3.3 Handling of missing observations

Check for missing observations in the dataset after smoothing and normalization.

```
sum(sum(ismissing(normalizedData)))
```

Find out how many customers have missing data.

```
missing_customers=sum(ismissing(normalizedData));
sum(missing_customers~=0)
```

Find customers with missing data.

```
missing_numbers = find(missing_customers==48)
```

Draw a graph of normalized data for one of the customers found.

```
plot(normalizedData(:,1403))
```

Remove customers with missing data.

```
cleanedData = normalizedData;
cleanedData(:, missing_numbers) = [];
```

4 Search for outliers (atypical) customers

4.1 Principal Components Analysis

Use the principal component analysis to find the main components for the analyzed data. Note - methods used from this point on require setting customers in rows rather than columns of the matrix.

```
data = cleanedData';
[rows, cols] = size(data)
[coef, score, latent] = pca(data);
```

Draw a graph showing how much information each of the components found contains (known as a scree plot).

```
plot(latent,'r-*')
```

Present all customers on a plane of two main components.

```
plot(score(:,1), score(:,2), 'o')
xlabel('1st principal component')
ylabel('2nd principal component')
```

4.2 DBSCAN algorithm

Points located in the low-density area are atypical customers. Use the DBSCAN algorithm to separate atypical customers from the rest. Before you run the DBSCAN algorithm, set its two key parameters: *minpts* and *epsilon*. Assume that the *minpts* parameter is to be at least *d+1*, where *d* is the length of the time series. In turn, to determine *epsilon*, use a graph showing the sorted Euclidean distances calculated for the two principal components from the *score* matrix.

```
minpts=50;
kD = pdist2(score(:,1:2),score(:,1:2),'euc','Smallest',minpts);
plot(sort(kD(end,:)));
grid
```

Using the graph created, take as *epsilon* the value from the vertical axis, from which the rapid increase of values begins.

```
epsilon=0.35;
```

Run the DBSCAN algorithm with the *minpts* and *epsilon* parameters fixed. The algorithm should assign each customer to one of two classes.

```
labels = dbscan(score(:,1:2), epsilon, minpts);
```

Present all customers on a plane of two main components with the two predetermined classes highlighted.

```
gscatter(score(:,1), score(:,2), labels);
```

Customers belonging to class -1 are outlier observations that should be discarded. Keep only customers with a label of 1.

```
non_outlier_data = data(labels==1,:);
```

Check how many customers remain after removing outliers.

```
[rows, cols] = size(non_outlier_data)
```

5 Hierarchical clustering

5.1 Calculation of distances between each pair of customers

Calculate the distance between each pair of customers. Use DTW (Dynamic Time Warping) distance, which is dedicated to time series. The result should be a sequence of numbers denoting the distances. You should get *w**(*w*-1)/2 numbers, where *w* is the number of customers.

```
distances_DTW = zeros(1,rows*(rows-1)/2);
counter=0;
for i=1:(rows-1)
    for j=(i+1):rows
        counter = counter+1;
        distances_DTW(counter) = dtw(non_outlier_data (i,:), ...
            non_outlier_data (j,:));
    end
end
```

5.2 Creating a hierarchy of clusters

Based on the calculated DTW distances, determine the cluster hierarchy. Compare the dendrograms created using different linkage methods: *ward, average, single, complete, weighted, median*.

```
Z1 = linkage(distances_DTW, "ward");
dendrogram(Z1,0)
Z2 = linkage(distances_DTW, "average");
dendrogram(Z2,0)
Z3 = linkage(distances_DTW, "single");
dendrogram(Z3,0)
Z4 = linkage(distances_DTW, "complete");
dendrogram(Z4,0)
Z5 = linkage(distances_DTW, "weighted");
dendrogram(Z5,0)
Z6 = linkage(distances_DTW, "median");
dendrogram(Z6,0)
```

5.3 Verification of the cluster hierarchy

Calculate the correlation between the distances calculated with DTW and the distances shown on the hierarchy tree. The calculated result will work as follows: the closer the value is to 1, the better the correspondence between DTW distances and distances from the tree.

```
c1 = cophenet(Z1, distances_DTW)
c2 = cophenet(Z2, distances_DTW)
c3 = cophenet(Z3, distances_DTW)
c4 = cophenet(Z4, distances_DTW)
c5 = cophenet(Z5, distances_DTW)
c6 = cophenet(Z6, distances_DTW)
```

5.4 Defining the clusters

For the cluster hierarchy obtained by Ward's method, define clusters by dissecting the dendrogram at the point of the longest branches.

```
dendrogram(Z1,0,'ColorThreshold',150)
clusters_H = cluster(Z1, "maxclust", 4)
```

5.5 Examination of the obtained clusters

Check how many customers belong to each of the defined clusters.

```
quantities = [sum(clusters_H==1),...
    sum(clusters_H==2),...
    sum(clusters_H==3),...
    sum(clusters_H==4)]
```

Calculate the average balance of customers in each cluster separately. When calculating the average, assume that account balances from all 48 months will be summed. Note - first, from the *customersclustering* matrix, remove the five customers who had missing observations after the normalization step, and also remove customers treated as atypical customers.

```
non_outlier_accbalance = customersclustering; % copying the balance data
non_outlier_accbalance(:, missing_numbers) = []; % removal of empty customers
non_outlier_accbalance(:, labels==-1) = []; % removal of outliers
avg_accbalance=[mean(sum(non_outlier_accbalance(:,clusters_H==1))),...
    mean(sum(non_outlier_accbalance(:,clusters_H==2))),...
    mean(sum(non_outlier_accbalance(:,clusters_H==3))),...
    mean(sum(non_outlier_accbalance(:,clusters_H==4))) ]
```

Calculate the standard deviation of the total customer account balance in each cluster separately.

```
std_accbalance=[std(sum(non_outlier_accbalance(:,clusters_H==1))),...
    std(sum(non_outlier_accbalance(:,clusters_H==2))),...
    std(sum(non_outlier_accbalance(:,clusters_H==3))),...
    std(sum(non_outlier_accbalance(:,clusters_H==4)))]
```

Draw a graph comparing the determined means and standard deviations in each cluster.

```
figure
bar([avg_accbalance ; std_accbalance]')
legend('mean','std')
```

Present the time series totals for each cluster on a single graph. The calculated totals should include the account balances of all customers belonging to a given cluster.

```
plot(sum(non_outlier_accbalance(:,clusters_H==1)'))
hold on
plot(sum(non_outlier_accbalance(:,clusters_H==2)'))
plot(sum(non_outlier_accbalance(:,clusters_H==3)'))
plot(sum(non_outlier_accbalance(:,clusters_H==4)'))
hold off
legend('cluster1','cluster2','cluster3','cluster4')
```

For comparison, show a graph of the total balance of all customers.

```
plot(sum(non_outlier_accbalance'))
```

Determine the centroid of each cluster. Assume that the centroid will be the so-called "averaged object", i.e. a vector whose values are equal to the average value of all objects (customers) in a given cluster. Note - this is not the average balance, but the average value of normalized data.

```
centroids(1,:)=mean(non_outlier_data(clusters_H==1,:));
centroids(2,:)=mean(non_outlier_data(clusters_H==2,:));
centroids(3,:)=mean(non_outlier_data(clusters_H==3,:));
centroids(4,:)=mean(non_outlier_data(clusters_H==4,:));
```

Depict all centroids on one common graph.

```
plot(centroids(1,:))
hold on
plot(centroids(2,:))
plot(centroids(3,:))
plot(centroids(4,:))
hold off
legend('c1','c2','c3','c4')
```

Create a separate chart for each defined cluster. On each chart, place the normalized balances of all customers belonging to that cluster and the centroid determined for that cluster. Mark the centroid with a bold red line.

```
plot(1:48,non_outlier_data(clusters_H==1,:), ...
    'Color',[.8 .8 .8],'LineWidth',0.1)
hold on
plot(centroids(1,:),'LineWidth',5,'Color','r')
hold off
plot(1:48,non_outlier_data(clusters_H==2,:), ...
    'Color',[.8 .8 .8],'LineWidth',0.1)
hold on
plot(centroids(2,:),'LineWidth',5,'Color','r')
hold off
plot(1:48,non_outlier_data(clusters_H==3,:), ...
    'Color',[.8 .8 .8],'LineWidth',0.1)
hold on
plot(centroids(3,:),'LineWidth',5,'Color','r')
hold off
plot(1:48,non_outlier_data(clusters_H==4,:), ...
    'Color',[.8 .8 .8],'LineWidth',0.1)
hold on
plot(centroids(4,:),'LineWidth',5,'Color','r')
hold off
```

Student assignments

1. Determine the centroid of all outlier customers that the DBSCAN algorithm has assigned to class -1.
2. Create a graph to show the centroids of each cluster together with the centroid of the outlier customers.
3. For each cluster, determine the minimum value from the average balances of all customers belonging to that cluster. Take the average of all 48 months for a given customer as the average balance. Then calculate the maximum value from the average balances and the median of the average balances in the same way.
4. For outlier customers, determine the values referred to in Task 3. Compare the results obtained with the four clusters studied.
5. Create a graph on which you place the normalized balances of all outlier customers and their centroid. Mark the centroid with a bold red line.
6. Save the original data on the account balances of the customers who were part of the clusters you created.