# Homework 2: Similarity measures on time series and graphs

Dr. Thomas Gumbsch
thomas.gumbsch@bsse.ethz.ch

Dr. Carlos Oliver
carlos.oliver@bsse.ethz.ch

Dr. Sarah Brüningk
sarah.brueningk@bsse.ethz.ch

Bowen Fan
bowen.fan@bsse.ethz.ch

Prof. Dr. Karsten Borgwardt
karsten.borgwardt@bsse.ethz.ch

*Submission deadline: 02.11.2022 at 08:00 am*

## Objectives

The goals of this homework are:

1. To learn how to model data as time series and graphs.
2. To implement similarity measures on time series and graphs.
3. To apply similarity measures on a heartbeat time series dataset and a molecular graphs dataset.
4. To explore theoretical aspects of these dissimilarity measures.

## Problem overview

In this homework you will implement the Dynamic Time Warping (`DTW`) distance function [5] on *time series* in Part I, and the Shortest Path Kernel (`SP`) similarity measure [1] on *graphs* in Part II. Additionally, you will be asked to address some theoretical questions about `DTW` and `SP`.

## Part I: Time Series

### Introduction

A time series is a sequence of measurements over a time interval. There are many methods to compute distances between time series. Dynamic Time Warping (`DTW`) is one

of the most famous and widely-used methods. In this homework, you will implement the DTW distance and use it on a heartbeat time series dataset.

Suppose we have two time series $t_1$ and $t_2$ of lengths $m$ and $n$, respectively. Given a maximum length $L$, we define an $(m,n)$-*warping path* to be a sequence $(w_l)_{l=1}^{L}$ of index pairs such that each term $w_l = (w_l^{(1)}, w_l^{(2)}) \in \{1,\dots,m\} \times \{1,\dots,n\}$ in the sequence indexes an element $t_1[w_l^{(1)}]$ of time series $t_1$ and an element $t_2[w_l^{(2)}]$ of time series $t_2$ for all $l = 1,\dots,L$. The length $L$ of the warping path satisfies $\max(m,n) \leq L < m+n-1$. An $(m,n)$-warping path can be understood as a sequence of correspondences between elements of $t_1$ and $t_2$, thus defining an alignment between the two time series. In DTW, the set of permissible $(m,n)$-warping paths is restricted to those which satisfy the following three properties:

**Boundary conditions:** The first and last terms of the warping path must be $w_1 = (1,1)$ and $w_L = (m,n)$, respectively. That is, the warping path must start comparing the first element of $t_1$ with the first element of $t_2$ and end comparing the last element of $t_1$ with the last element of $t_2$.

**Monotonicity:** Let $w_l = (i,j)$ and $w_{l+1} = (i',j')$ be two consecutive elements in the warping path. Then, $i \leq i'$ and $j \leq j'$ must hold. That is, the warping path cannot go 'backwards'.

**Unit-size steps:** Let $w_l = (i,j)$ and $w_{l+1} = (i',j')$ be two consecutive elements in the warping path. Then, $i' - i \leq 1$ and $j' - j \leq 1$ must hold, with at least one of them being a strict inequality. That is, if $w_l = (i,j)$, the next element in the warping path $w_{l+1}$ must be one of $(i+1,j)$, $(i,j+1)$, or $(i+1,j+1)$.

If we denote by $\mathcal{W}(m,n)$ the set of all $(m,n)$-warping paths satisfying all three conditions above, the DTW distance between two time series $t_1$ and $t_2$ of lengths $m$ and $n$ is defined by the following optimisation problem:

$$\mathrm{DTW}_d(t_1,t_2) := \min_{w \in \mathcal{W}(m,n)} \sum_{l=1}^{L(w)} d\left(t_1[w_l^{(1)}], t_2[w_l^{(2)}]\right), \tag{1}$$

where $d\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ is an element-wise, user-defined distance function.

$\mathrm{DTW}_d(t_1,t_2)$ can be efficiently calculated using *dynamic programming*, a technique for creating solutions to optimisation problems incrementally. As you learned in the lecture, DTW is perfectly suited for this task because it is defined recursively. Define $C$ to be a $(m+1) \times (n+1)$ matrix such that $C_{i+1,j+1}$ equals the DTW distance between the time series $t_1[1:i]$ containing the first $i$ elements of time series $t_1$ and the time series $t_2[1:j]$ containing the first $j$ elements of time series $t_2$. Each entry $C_{i,j}$ can be obtained in $\mathcal{O}(1)$ time provided the values of $C_{i-1,j-1}$, $C_{i,j-1}$ and $C_{i-1,j}$ are known, by calculating

$$C_{i,j} = d(t_1[i], t_2[j]) + \min\left(C_{i-1,j-1}, C_{i,j-1}, C_{i-1,j}\right). \tag{2}$$

With $C$ defined in such a way, we have $\mathrm{DTW}_d(t_1,t_2) = C_{m+1,n+1}$. Note that if $i = 1$, then $C_{i-1,j-1}$ and $C_{i-1,j}$ are undefined and $C_{i,j} = d(t_1[i], t_2[j]) + C_{i,j-1}$. Analogously, if $j = 1$, then $C_{i,j} = d(t_1[i], t_2[j]) + C_{i-1,j}$. In order to avoid explicitly dealing with these particular cases, we define so-called *sentinel values*

$$\begin{aligned} C_{0,0} &= 0, \\ C_{i,0} &= \infty \quad \text{for } i = 1,\dots,m \\ C_{0,j} &= \infty \quad \text{for } j = 1,\dots,n \end{aligned} \tag{3}$$

to ensure render that Equation (2) is valid for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

## Dataset

You will work on the 'ECG' heartbeat time series data from the UCR time series archive [2], that has been widely used as one of the benchmark datasets for evaluating time series classification. Each record of 'ECG' is a time series of measurements recorded by one electrode during one heartbeat. The data has been annotated by a cardiologist and a label of normal or abnormal is assigned to each data record. In Exercise 1, you will work on the dataset stored in `ECG200_TRAIN.txt`. There are 100 time series samples and the length of each sample is 96 (same length for all samples). The first column indicates the class of samples, where $-1$ means the abnormal heartbeat and 1 the normal heartbeat. The other 96 columns are records of time series samples.

## Exercise 1

**Exercise 1.a** A common variant of DTW adds an additional requirement in the definition of valid $(m, n)$-warping paths, namely:

**$w$-constrained warping:** Any element $w_l = (i, j)$ in the warping path must satisfy $|i - j| \leq w$, where $w \geq |m - n|$ is a user-defined hyperparameter.

Create a Python function named for implementing the `DTW` distance function with $w$-constrained warping:

```
constrained_dtw(t1, t2, w)
```

**Please note**:

- The first two arguments of the `constrained_dtw` function, namely `t1` and `t2`, are both lists of type `float`, while the third argument `w` is a non-negative integer not smaller than the difference in length between the two time series `t1` and `t2` to be compared.
- The name of the function and its arguments should match the above description.
- The return value is a `float`.
- Use the Manhattan distance $d(x, y) = |x - y|$ as the base elementwise distance for DTW, that is, $d(t_1[i], t_2[j]) = |t_1[i] - t_2[j]|$.
- **Do not use libraries or external source code that implement the `DTW` distance function, with or without $w$-constrained warping**. The purpose of this exercise is for you to understand how to implement this measure.

Since the lengths of different time series samples in the 'ECG' dataset are the same, they can also be treated as vectors. In such a case, distance functions on vectors can also be used to compare time series. In this exercise, you will exploit this to compare DTW with the `Manhattan` distance on vectors.

Extend the provided skeleton with your implementations of the Manhattan distance or create a Python script named `compute_dtw.py` to read the `ECG200_TRAIN.txt` dataset and compute the average distances between groups using:

i) The `Manhattan` distance, treating the time series as vectors.

ii) The `DTW` distance with $w$-constrained warping and the Manhattan distance as the elementwise metric, for values of $w$ in $\{0, 10, 25, \infty\}$.

**Please note**:

- Your program `compute_dtw.py` will receive 2 command line arguments:

  `--datadir path`: the path to the directory where the input file is stored.

  `--outdir path`: the path to the directory where the output file will be stored.

- The program will compute the average `DTW` and `Manhattan` distances between groups and output them to the file `timeseries_output.txt`. Its format is shown in Figure 1. Your output file should include the header as well and output rows and columns in the specified order. To store the columns in the specified order, you can use a snippet like this one (if you are not using our pre-defined script):

```
    w_vals = [0, 10, 25, float('inf')]

    # Write header for output file, using tab-separated columns
    f_out.write('{}\t{}\t{}\n'.format(
      'Pair of classes',
      'Manhattan',
      '\t'.join(['DTW, w = {}'.format(w) for w in w_vals]))
    )

```

  This ensures that individual columns are separated by tabs (tabulators), simplifying the correction of the exercise later on.

- If you write your own script without relying on our provided skeleton, make sure that you used the name `compute_dtw.py` for your script. This will simplify grading.

| Pair of classes | Manhattan | DTW, w = 0 | DTW, w = 10 | DTW, w = 25 | DTW, w = inf |
|---|---|---|---|---|---|
| abnormal:abnormal | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 |
| abnormal:normal | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 |
| normal:normal | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 |

Figure 1: Format of output file `timeseries_output.txt`.

**Exercise 1.b** Compare and discuss the results of the `DTW` and `Manhattan` distances on separating abnormal and normal heartbeats. Which one would you choose in practice and why?

**Exercise 1.c** Discuss the effect that hyperparameter $w$ has on the DTW distance and its ability to separate abnormal and normal heartbeats.

**Exercise 1.d** Is the DTW distance a metric? If not, give examples showing which conditions are not satisfied.

**Exercise 1.e** What is the runtime complexity of computing the DTW distance with $w$-constrained warping? You may assume that $m = n$. By contrast, what is the runtime complexity of DTW (without any constraints)?

# Homework Part 2: Graphs

## Introduction

Graphs provide a flexible way to represent complex data, such as molecules. As a result, they play a central role in data mining. Firstly, we consider the similarity between nodes in the graph. The most popular distance function is the length of the shortest path between nodes. Secondly, we compute the similarity between graphs. Graph kernels have been proposed in order to do this job. The Shortest Path kernel (SP) is one that uses shortest paths to compare graphs.

Suppose we have two graphs G1 with m vertices and G2 with n vertices (for simplicity, in this homework, we only consider unweighted and undirected graphs). The corresponding adjacency matrices are A1 and A2, respectively. Then the shortest path kernel (SP) between G1 and G2 can be calculated by the following steps:

1. Transform the adjacency matrix A1 and A2 into shortest-path matrices S1 and S2, where S(i,j) is the length of shortest path between node i and j.
2. The SP kernel is calculated using S1 and S2:

$$K_{\text{sp}}(S1, S2) = \sum_{e_1 \in S1} \sum_{e_2 \in S2} K^1_{\text{walk}}(e_1, e_2), \tag{4}$$

where

$$K^1_{\text{walk}}(e_1, e_2) = \begin{cases} 1, & \text{if weight}(e_1) = \text{weight}(e_2) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

3. Note that $e_1$ can be seen as an edge walk of length 1 in S1, which is a non-zero entry of S1. Since S1 is symmetric, only the upper or lower triangular matrix of S1 should be considered.

## Dataset

You will work on the 'MUTAG' dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds [3]. As stated in the dataset description page: *'...Sample graphs are labeled according to whether or not they have a mutagenic effect on the Gram-negative bacterium Salmonella typhimurium.'*[1]

Graphs with a mutagenic effect are labelled 1, all others are labelled -1. Each sample graph is represented by its adjacency matrix. The adjacency matrices and labels are stored in MUTAG.mat. This dataset will be used in Exercise 2. Instructions for how to load this file in Python will be introduced later in Exercise 2.b.

[1] See https://www.bsse.ethz.ch/mlcb/research/machine-learning/graph-kernels.html for more details.

## Exercise 2

**Exercise 2.a** Create a Python script named `shortest_path_kernel.py` in which you will implement (i) Floyd-Warshall's algorithm [4] to compute shortest path lengths and (ii) `SPKernel`. The pseudo-code of Floyd-Warshall's algorithm can be found in the lecture slides *Part 1: Introduction and Distance Functions*. Use the following names (and parameters) for the functions (as provided in the skeleton file `shortest_path_kernel.py`):

```
floyd_warshall(A)
sp_kernel(S1, S2)
```

**Please note**:

- The argument of the function `floyd_warshall` is the adjacency matrix `A` of a graph. If there is no link between two nodes `i` and `j`, then `A(i,j)` will be 0. For such pairs of nodes `i` and `j`, the shortest distance should be initialized as infinite. Therefore, in practice, you will need to convert all the zero entries except the diagonal of `A` to a very large value before calculating the shortest path lengths.
- The return value of `floyd_warshall` is the shortest-path matrix `S` of a graph.
- The parameters of the function `sp_kernel` are two shortest-path matrices `S1` and `S2`.
- The return value of `sp_kernel` is a `float`.
- **Do not use libraries or external source code that implement either** `sp_kernel` **or** `floyd_warshall`. The purpose of this exercise is for you to learn how to implement the shortest-path kernel.

**Exercise 2.b** Create a script named `compute_sp_kernel.py` to read the `MUTAG.mat` dataset and compute the average `SP` kernel similarities between the abnormal and normal class. You have to `import` your own functions from `shortest_path_kernel` by issuing

```
from shortest_path_kernel import floyd_warshall
from shortest_path_kernel import sp_kernel
```

at the beginning of your script. Using the module `scipy.io.loadmat`, the adjacency matrices and label information can be loaded using the following snippet:

```
import scipy.io
mat = scipy.io.loadmat('PATH to MUTAG.mat')
label = np.reshape(mat['lmutag'], (len(mat['lmutag'], )))
data = np.reshape(mat['MUTAG']['am'], (len(label), ))
```

Afterwards `data` will contain a set of adjacency matrices. You can index these matrices individually by iterating over the first dimension of the array, i.e.

```
  for matrix in data:
    # Do something exciting with the adjacency matrix!
    pass
```

This snippet requires the `scipy` module to function properly.

**Please note**:

- Your program `compute_sp_kernel.py` will receive 2 command line arguments:

  `--datadir path`: the path to the directory where the input file is stored.

  `--outdir path`: the path to the directory where the output file will be stored.

- The program will compute the average similarities and output them to a file named `graphs_output.txt`. Its format is shown in Figure 2. Similar to the previous exercise, please ensure that columns are tab-separated.

| Pair of classes | SP |
|---|---|
| mutagenic:mutagenic | 99.9 |
| mutagenic:non-mutagenic | 99.9 |
| non-mutagenic:non-mutagenic | 99.9 |

Figure 2: Format of output file `graphs_output.txt` file.

**Exercise 2.c** What is the runtime complexity of Floyd–Warshall's algorithm? What is the runtime complexity of `SP`? How would you improve the runtime?

# Grading and submission guidelines

This homework is worth a total of 100 points. Table 1 shows the points assigned to each exercise.

Table 1: Grading key for Homework 2

| | | |
|---|---|---|
| **50 pts.** | **Exercise 1** | |
| | 30 pts. | Exercise 1.a |
| | 5 pts. | Exercise 1.b |
| | 5 pts. | Exercise 1.c |
| | 5 pts. | Exercise 1.d |
| | 5 pts. | Exercise 1.e |
| **50 pts.** | **Exercise 2** | |
| | 20 pts. | Exercise 2.a |
| | 20 pts. | Exercise 2.b |
| | 10 pts. | Exercise 2.c |

# Acknowledgements

This exercise sheet was created by Xiao He, Dean Bodenham, Dominik Grimm, Damian Roqueiro, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt.

# References

[1] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[2] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[3] A. Debnath, R. Lopez de Comadre, G. Debnath, A. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, pages 786–797, Feb 1991.

[4] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[5] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics Speech and Signal Processing*, 26(1): 43–49, 1978.