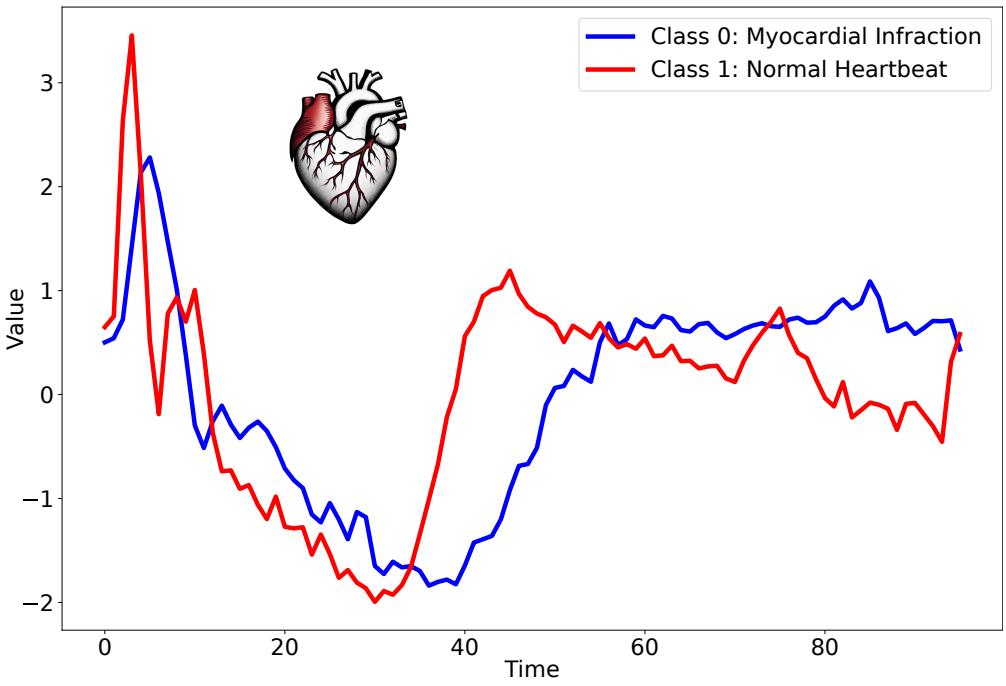
TD2: Time Series Analysis

Introduction to Data Science en Python - ENSISA CPB2

Ali El Hadi ISMAIL FAWAZ

November 14, 2024

1 Introduction



Time series are a particular type of data that represent sequences of ordered values over time. The data can be timestamped explicitly (a date is given for each value) or implicitly (the order of data occurrence is considered).

These data can, for example, represent the evolution of values provided by a sensor over time (such as temperature or pressure), an electrocardiogram, or even sound.

The classification of time series involves assigning a class to a data point based on its characteristics. For example, electrocardiograms of patients can be classified as "normal" or "abnormal" based on their characteristics.

In this context, the nearest neighbor algorithm is often used to classify time series. Given a time series to be classified, this algorithm involves finding the "nearest" time series in a set of time series with known classes. Thus, there are two sets of data: a first set, called "train," composed of time series with known classes, and a second set, called "test," for which we seek to predict the class.

Example of a time series sample from an ECG signal while monitoring the heart can be seen in the figure above. Example is taken from the ECG200 dataset of the UCR archive [1]. The ECG200 contains two set of classes, differentiating if the patient has a heart issue or not.

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

2 Reading the Data

For your first task, you will be asked to implement a function that reads a dataset in its csv form and return four different elements:

- `xtrain`, a numpy array of shape (number of train samples, length of time series) containing the training samples.
- `ytrain`, a numpy array of shape (number of samples) containing the labels of each sample in `xtrain`
- `xtest`, a numpy array of shape (number of test samples, length of time series) containing the testing samples.
- `ytest`, a numpy array of shape (number of test samples) containing the labels of each sample in `xtest`

The data are provided as follows. For each dataset, their exist a folder with the name of the dataset. Inside each folder, their exist two `.tsv` files, `TRAIN.tsv` and `TEST.tsv`. Each of the two `.tsv` files are organized as follows: each line is one time series example, the columns are the time indices (similar to what we call attributes in tabular data such as the Pokemon data). The first column however contains the label of the time series example, so make sure to extract this column for `ytrain` and `ytest`.

Here is the skeleton of your code:

```
import numpy as np

def load_data(file_name):
    train = np.loadtxt(file_name+"/TRAIN.tsv")
    test = np.loadtxt(file_name+"/TEST.tsv")

    # Your code here
    ...

    return xtrain, ytrain, xtest, ytest

# Usage of the function
xtrain, ytrain, xtest, ytest = load_data(file_name="ECG200")
```

3 Normalizing your Data

In the case of time series datasets, we tend to do a normalization technique called Z-normalization:

$$x = \frac{x - \mu_x}{\sigma_x}$$

where, x is a time series of any length, μ_x is the mean value of x over time and σ_x is the standard deviation of x over time as well:

```
import numpy as np

mu_x = np.mean(x)
sigma_x = np.std(x)
```

Your task now is to implement the following function that takes `xtrain`, `xtest` and return a znormalized version of them.

```
def znormalize(xtrain, xtest):
    # Your code here
    ...

    return xtrain_znormalized, xtest_znormalized

# Usage of the function
xtrain, xtest = znormalize(xtrain, xtest)
```

4 Plotting the Data

Now you will implement a function that plots the time series data with a different color for each class. Your plot should be made of two subplots, on for each of train and test samples. Think about reading about the documentation of `plt.subplots` in the `matplotlib` package. Your code should save the figure at the end in the `.pdf` form instead of using `plt.show`. The name of the file should be the same as the name of the dataset.

Hint: use `np.unique` to get the unique classes.

```
import numpy as np
import matplotlib.pyplot as plt

def plot_data(file_name):

    xtrain, ytrain, xtest, ytest = load_data(file_name=file_name)

    xtrain, xtest = znormalize(xtrain, xtest)

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20,10))

    # Your code here
    ...

    fig.savefig(file_name+'/'+file_name+'.pdf', bbox_inches="tight")
```

5 Similarity Measures

We need to define the similarity between time series in order to define what a neighbor is. Here your task will be to implement functions that take two input time series `x` and

`y`, two `numpy` arrays of shape (length of time series). Below we define two know similarity measures used for time series, your task is to implement a function for each similarity measure.

5.1 Euclidean Distance

The most intuitive approach to compare two time series is using the Euclidean Distance defined below between two time series of length L:

$$ED(x, y) = \sqrt{\sum_{i=0}^{L-1}(x[i] - y[i])^2}$$

This similarity measure supposes a perfect temporal alignment between the time series, which is not the case always.

5.2 Dynamic Time Warping (DTW)

DTW is a similarity measure specific for time series data, because it takes into consideration that the two time series are not perfectly aligned on the time axis.

$$DTW(x, y) = \min_{\pi} \sqrt{\sum_{(i,j)\in\pi} (x[i] - y[j])^2}$$

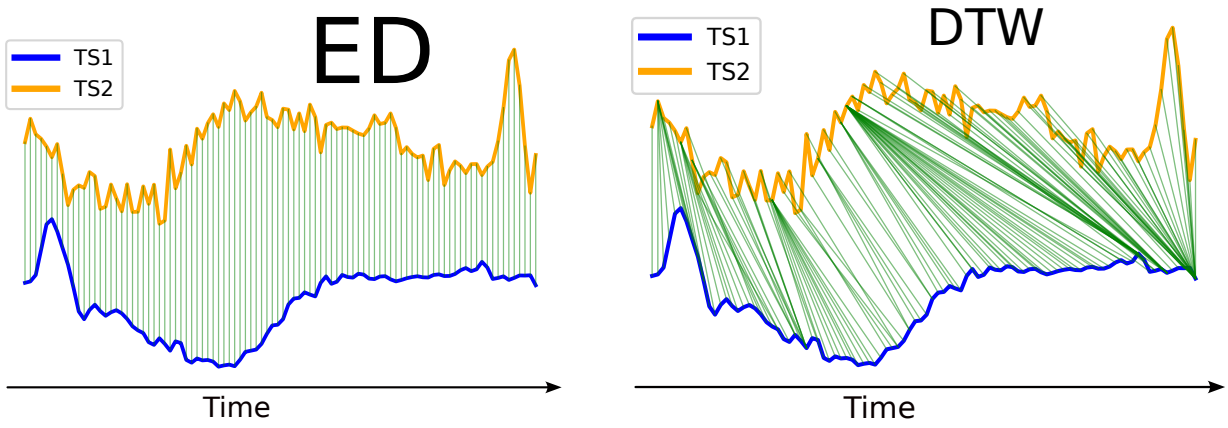
where π is the alignement path.

DTW works on finding the optimal alignment path between two time series by first generating a squared difference metrics, see Algorithm 1 for more info.

Algorithm 1: Dynamic Time Warping (DTW)

Data: Time Series x, Time Series y, both of length L
Result: DTW distance

Initialize a matrix D of size $L + 1 \times L + 1$;
Set $D[0, 0] = 0$;
Set $D[:, 0] = \infty$;
Set $D[0, :] = \infty$;
for $i \leftarrow 1$ **to** L **do**
    for $j \leftarrow 1$ **to** L **do**
        Set $d = (x[i] - y[j])^2$;
        Set $D[i][j] = d + \min(D[i - 1, j], D[i, j - 1], D[i - 1, j - 1])$;
    end
end
return $D[L, L]$;



We can see in the figure above how DTW finds the optimal alighment path between the two series.

```
import numpy as np

def euclidean_distance(x,y):
    # Your code here
    ...

def dynamic_time_warping(x,y):
    # Your code here
    ...

# Usage of code

xtrain, ytrain, xtest, ytest = load_data(file_name="ECG200")

xtrain, xtest = znormalize(xtrain, xtest)

x = xtrain[0]
y = xtest[0]

print("euclidean distance between x and y:", euclidean_distance(x,y))
print("dynamic time warping between x and y:", dynamic_time_warping(x,y))
```

6 Nearest Neighbor Classifier

In this question, your task is to implement a Nearest Neighbor Classifier where we will find the similarity measure between all samples in `xtest` with all samples in `xtrain`, for each sample in `xtest` we will affect a label equal to the label of its neighbor. We define neighbor as the time series from `xtrain` having the smallest similarity measure with the time series in `xtest`. The function `nearest_neighbor` should take as argument `xtrain`, `ytrain`, `xtest` and whcih `similariy_measure` to use, either "euclidean distance" or "dynamic time warping". The function should return `ypred` a numpy of shape (number

of test samples) containing the predicted labels.

```
import numpy as np

def nearest_neighbor(xtrain, ytrain, xtest, similarity_measure):

    # Your code here
    ...

    return ypred
```

7 Evaluation Metric

To evaluate how good our nearest neighbor classifier performed, we will use the accuracy metric that counts the number of times our prediction was correct:

$$accuracy(ytest, ypred) = \frac{1}{len(ytest)} \sum_{i=0}^{len(ytest)-1} (ypred[i] == ytest[i])$$

Your task is to implement this accuracy metric and use it on the nearest neighbor classifier on 5 datasets: ECG200, Coffee, ItalyPowerDemand, SyntheticControl and TwoLeadECG. Your results should be the same as the ones in[2].

8 Using a Package: aeon

Start by installing aeon using :

```
pip install aeon
```

The aeon[3] package contains the nearest neighbor classifier that you need:

```
from aeon.classification.distance_based import KNeighborsTimeSeriesClassifier as KNN

file_name = "ECG200"

xtrain, ytrain, xtest, ytest = load_data(file_name=file_name)

xtrain, xtest = znormalize(xtrain, xtest)

# Continue your code here to use aeon to print the accuracy score of KNN using euclidean
    distance and dtw. Check the documentation of aeon
...
```

[2] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
[3] https://www.aeon-toolkit.org/en/latest/index.html