Dynamic Time Warping for Sequence-to-Sequence Matching

String and Sequence Matching problems refer to problems in which the sequences to be compared can be of different length. Aligning them is an intrinsic ingredient of the solution. The matching measure or distance is an accumulation of so called local distances that can be computed on the elements of the sequences.

The well known **Levenshtein** algorithm is commonly used when the sequence elements are atoms drawn from a final alphabet, such as the letter of the alphabet in string comparison.

When the sequences are made up of real numbers or vectors and the sequence is time based, we tend to refer to **Dynamic Time Warping** as in speech applications.

These two algorithms are intrinsically the same and may generally speaking be referred to as implementations of the **Dynamic Programming Principle**. Obviously the local distance measure is different when operating on atoms or continuous vectors. Also the transition constraints may be application dependent.

In this tutorial we use **DTW** as a generic name; also because we use the *pyspch.dtw* module for the computations.

DTW finds a sequence distance between two sequences data

$$X_{i=1:N} \text{ and } Y_{j=1:M}$$

The **sequence distance** is the minimum of the cummulative distances amongst all possible alignment paths

$$(X_{i(s)}, Y_{j(s)}) \text{ for } s = 1 \ldots L$$

whereas the cummulative distance along a given path is defined as:

$$\sum_s D_l\left(X_{i(s)}, Y_{j(s)}\right) + D_t\left(trans\right)$$

with: $trans = (i(s) - i(s-1), j(s) - j(s-1))$

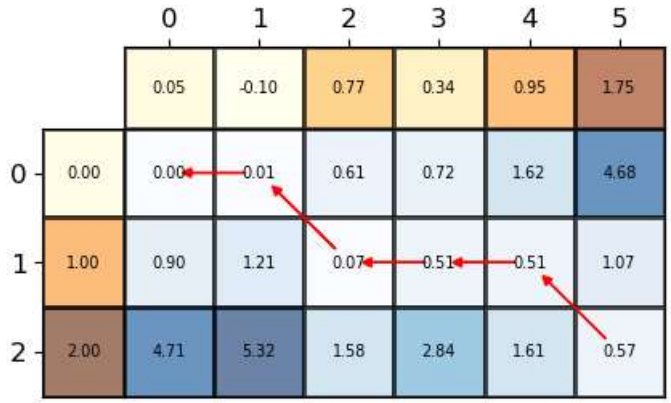
- with $D_l(X, Y)$ denoting a *local distance measure* that measures distance between 2 feature vectors
- with $D_t(..)$ denoting a *transition cost* depending on the transition that was taken
- boundary constraints:
  - if you need $\{X\}$ to be a match for $\{Y\}$ then the alignment should start in the origin $(1, 1)$ and end in (N, M)
  - if you want to find the best possible subsequence match of $\{X\}$ in $\{Y\}$, then any starting point (1,j_1) and any endig point (N,j_L) is acceptable
- certain transition constraints are generic:
  - i and j need to *progress* in time, i.e. $i(s+1) >= i(s)$
  - the diagonal transition (+1,+1) is natural and by default included in the list of allowed transitions
- certain transition constraints are specific to the applied transition model:
  - In a Levenshtein model for string matching
    - DIAGONAL: (+1,+1), INSERTION: (+1,0), DELETION: (0,+1)
    - For INSERTION and DELETION only a fixed transition cost is added
    - For DIAGONAL transitions the local distance cost is added
  - In (vanilla) DTW
    - the basic transitions are: (+1,+1), (+1,0), (0,+1)
    - a local distance metric is defined, allowing to built the local distance matrix $D_l\left(X_i, Y_j\right)$
    - either no transition costs are taken into account, or a transition penalty is added to the non diagonal transitions
  - In Symmetric Itakura
    - the possible transitions are: (+1,+1), (+2,+1), (+1,+2)
    - a local distance matrix is built as above
    - the long moves may be seen as double transitions, therefore the local distances 'along the way' should be accumulated; alternatively a multiplicative cost (eg. 2) can be implemented

The two DTW models described above, can be visualized as here: dtw_constraints

Example 1: 1D feature vectors

- first define two some sequences
  The feature vectors are scalars (1D vectors)
- run a standard DTW algorithm on it and print a trellis
  The local distance metric applied is 'Euclidean Squared' and use the Levenshtein algorithm for aligning, i.e. we can move diagonal, horizontal or vertical (always in the direction of time),

```
DTW distance: 0.573
```

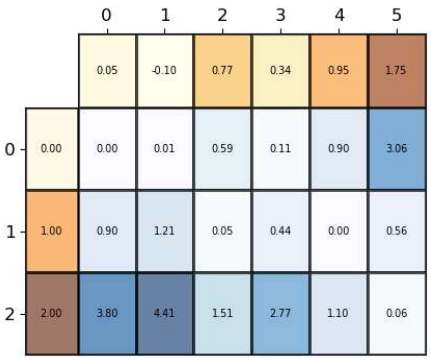


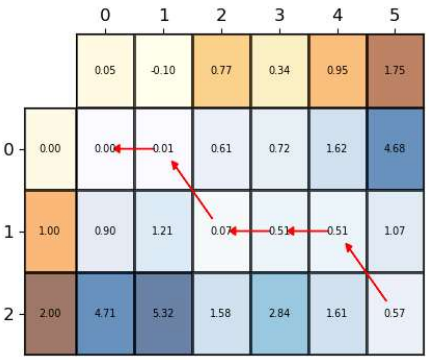
Steps in the DTW algorithm

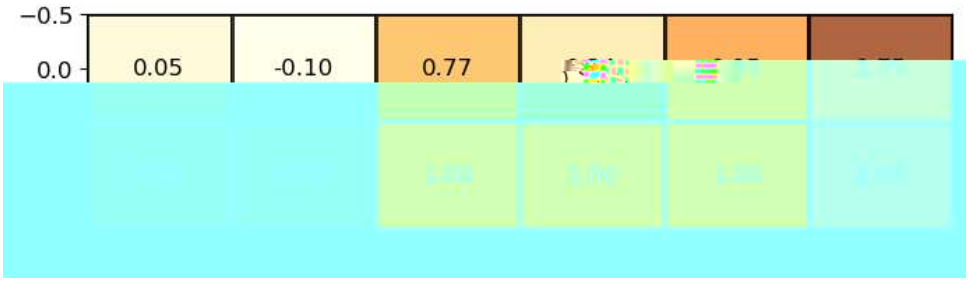
1. Local Distance Computation Compute for each pair (x_i, y_j) a distance. This feature by feature distance, organized in a matrix is called the "local distance matrix"
2. Trellis Computation In a trellis we compute accumulated distances. Each cell contains a sum of local distances , starting up to the current cell. Thus cell (Nx,Ny) contains the full sequence distance.
   Obviously the computation of cummulative distance this is subject to the allowed transitions.
   To maintain the memory of how we moved from one cell to another we add backpointers in each cell
3. Backtracking In order to find the alignment between the two sequences. We follow backpointers from the end back to the start

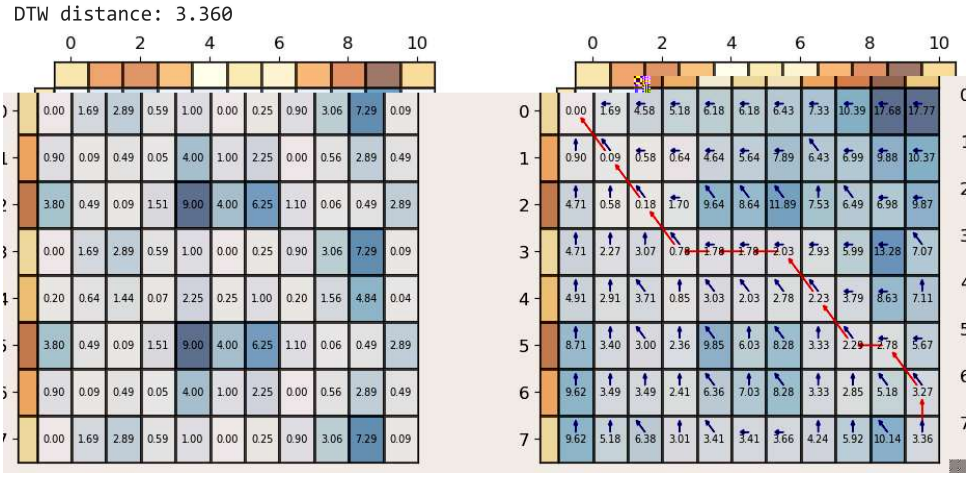
Trellis Representation

The trellis is a matrix arrangement in which each cell (i,j) says something about the relationship between x[i] and y[j] or up to that moment in time for both sequences.
We tend to add extra space in the plot for the feature vectors, typically clearly visable by the different colormaps.



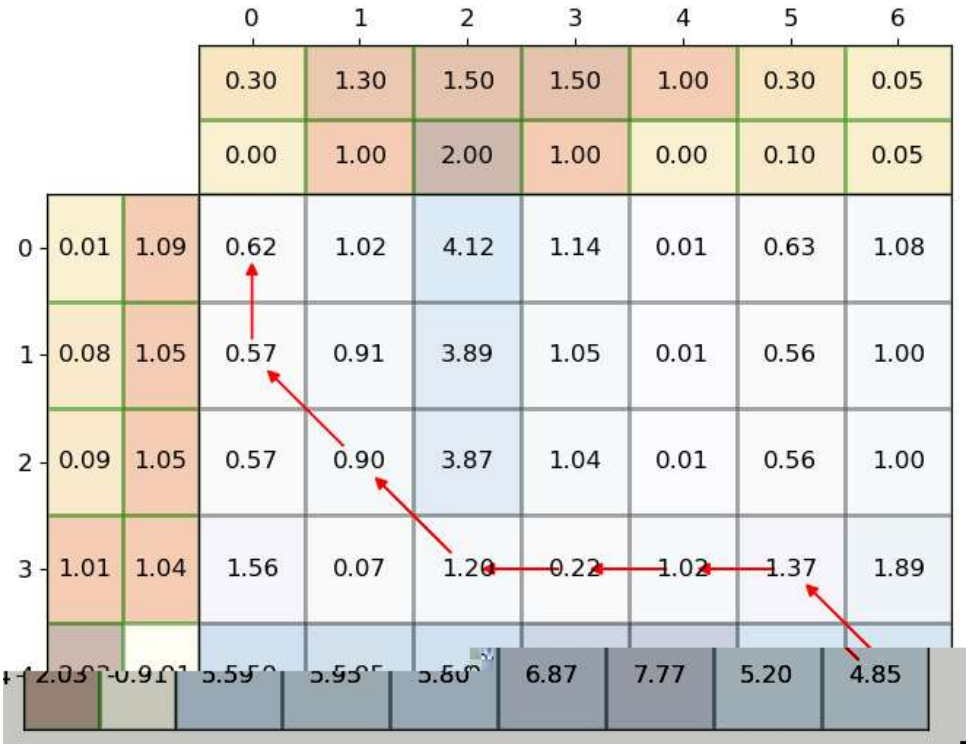


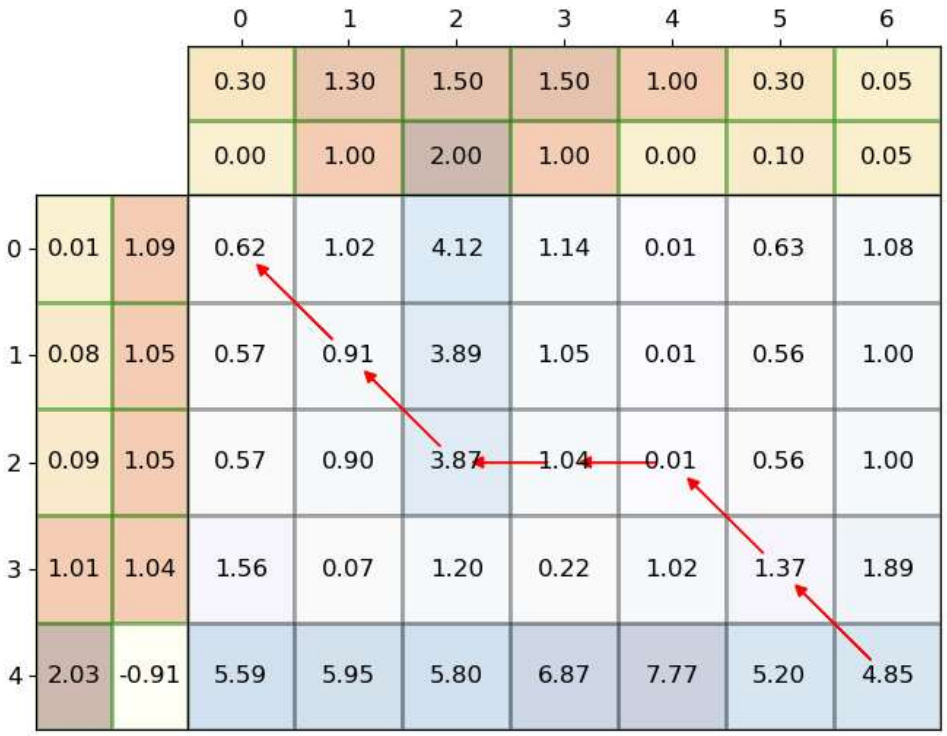




Multi-dimensional Features

In this exampe we pass 2D feature vectors





ITAKURA constraints

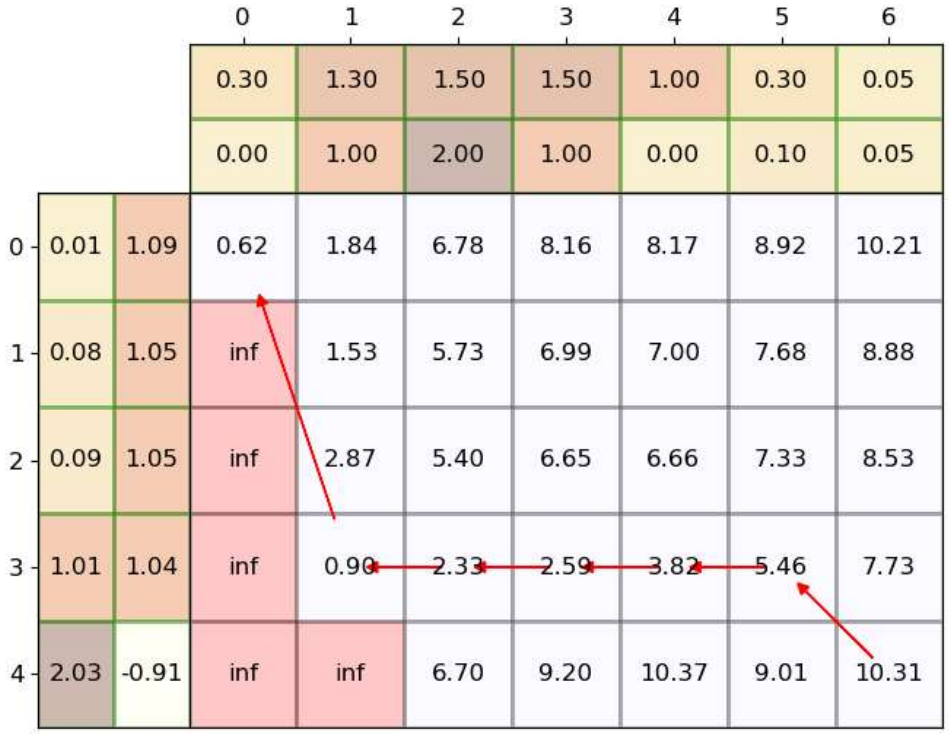
In sequence matching one may generally expect that the alignment path will follow the main diagonal closely. Long verticle or horizontal segments in the alignment are indications of a bad mismatch or significant insertion/deletions at those locations.

In the case of speech, things are no different as speaking rate differences tend to be limited. Hence, popular in the speech community are the so called ITAKURA constraints in sequence matching.
Itakura allows single jump in x or y direction while the other index just increases by 1 .
In the dtw.dtw() implementation we penalize such jump moves by 20% vs. regular diagonal moves.
This allows for a speedup of x2 on either axis. At the same time it implies that certain cells in the trellis can never be reached.
One of the bigger problems is that it is very hard to align variable length noise segments as for these a warping >> 2 may be acceptable.

In the dtw() module a few alternatives are available, that may be selected depending on the application:

- ITA: baseline symmetric DP with single jump allowed in x or y
- ITX: x-time synchronous DP
- ITY: y-time synchronous DP
- ITX3: x-time synchronous DP with Delta-y up to 3
- ITY3: y-time synchronous DP with Delta-x up to 3
- ITS: symmetric DP with infinite sized warping in both x and y possible (combintation of 'ITA' and 'DTW').

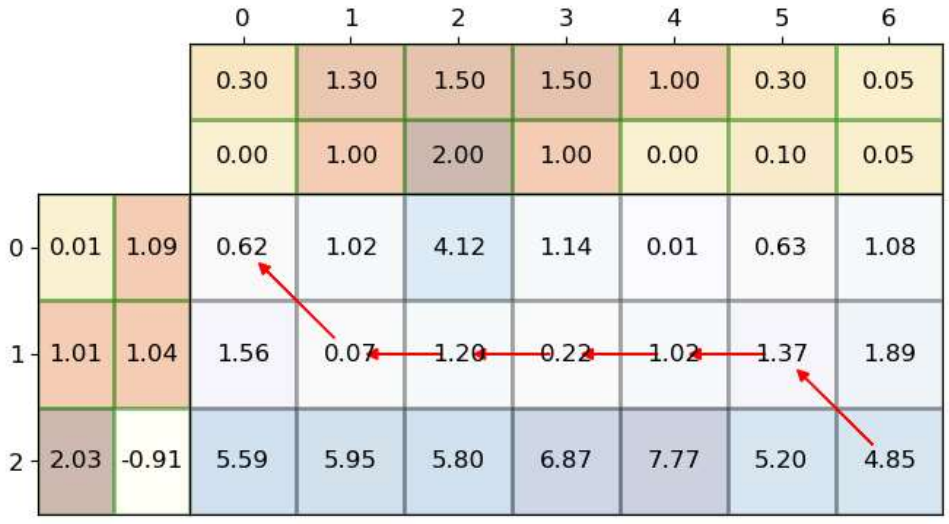
Due to the warping constraints, certain areas in the trellis are not reachable and show infinite cost. Exception is the ITS variant in which exery cell in the trellis is reachable




```
DTW distance =  10.307493819563167
```

Admissible Transitions and acceptable speedups

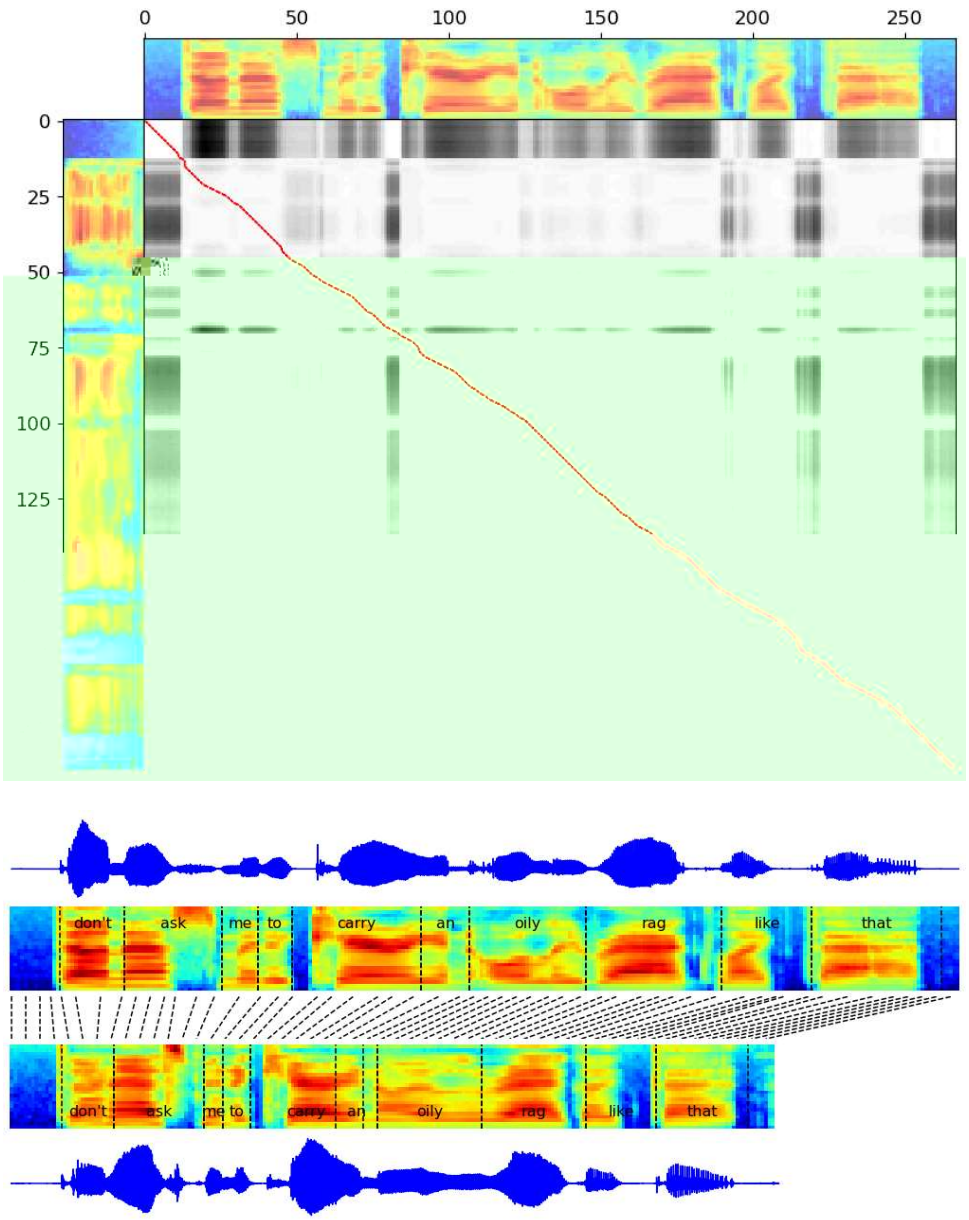
The ITAKURA constraints allow for a maximum x2 speedup in either direction.
This makes sense during speech, but in silence periods much more may be required.
Therefore it is advised to combine endpointing with the ITAKURA DTW implementation. If Nx > 2 times Ny, then the final cell in the trellis will not be reached. The **IT2** constraints are a mix of ITAKURA and LEVENSHTEIN but with high costs on the horizontal / vertical moves. It will resut in an alignment for ANY lengths Nx,Ny and in matching segments the ITAKURA constraints will dominate.




```
DTW distance =  9.338238609939435
```

Now with speech data

DTW


0
50
100
150
200
250
0
25
50
75
100
125

don't
ask
me
to
carry
an
oily
rag
like
that
don't
ask
me to
carry
an
oily
rag
like
that