Part 1b (written)

Describe DTW

Dynamic Time Wrapping is an algorithm that determines how similar two time series are by estimating the optimal alignment between them. The two time series can vary in time and speed, which means one may be shorter than the other in time length, and one's change may be occur faster or slower than the other. Since the two time series' lengths are different and changes may occur at different time, in order to compare them, we need to stretch them. In other word wrap one or both of them. If illustrating this on a graph, it can be think of that a point in time sequence A is linked with number of points in time sequence B.

The goal of the DTW algorithm is to calculate a warping path to align them in which the path minimize the cumulative distance between the two time series. To find the wrapping path, we first computing a cost matrix and then find a warping path (optimal alignment path) using the cost matrix.

Explain how DTW differs from simple sequence matching, highlight the unique aspects of DTW that allow it to handle variation in speed and timing

Dynamic Time Wrapping differs from simple sequence matching through its ability to stretching and compressing the time series, hence enable it to compare two time series that are different in speed and length. Simple sequence matching usually takes a one to one relationship in comparing two time series, for example, calculating the distance between the first point in the first time sequence with the first point in the second time sequence and so on, this is a linear comparison in which one point align with another, and this simple approach has issue with outlier points and is incapable in handing time series with changes occurred at different time. Whereas the DTW allows alignment for one point in sequence A with multiple points in sequence B, this means it can do non-linear alignments between sequences, and essentially, stretching the time sequences to do a more accurate comparison.

Detail the initial setup of the cost matrix: describe what each cell represents and how the initial value are set.

To initialise the cost matrix, we need to first create a matrix with n+1 rows and m+1 columns in which n is the size of the time sequence A and m is the size of the time sequence B, in c, the matrix would be represented as a long double 2-dimensions array. Then we initialise all the cells to have an infinity value and the infinity value for a data type is an estimated value which may differ in different computing system, we then set the value for the cell(0, 0) of the matrix to be zero.

A cell at position (i, j) in the cost matrix represents the minimum cumulative cost of aligning the ith element of sequence A with the jth element of sequence B. The cell at (0,0) is set to be zero, as this position represents the cumulative cost of aligning the 0^{th} element of sequence A with the 0^{th} element of sequence B, but there is no elements being aligned at this point, so we set it to zero.

Describe the process of calculating the optimal path through the cost matrix: explain the criteria used to determine the path (minimise total cost)

We calculate the optimal path by populating the cost matrix, we fill each cell in the matrix except for the cells in the first row and the first column with the minimum cumulative cost in aligning ith element in sequence A with the jth element in sequence B. The minimum cumulative cost is calculated by finding the absolute value of subtracting the (j-1)th point in the sequence B from (i-1)th point in the sequence A, and adding that absolute value to the cost of reaching the focus cell (i,j) from its neighbour cell, which is the minimum value among costs of three cells (the one on the left of our focus point, the one on the top of our focus point and the one on the top left of our focus point). We fill the cells row by row, from top to bottom.

The formula for finding minimum cumulative cost for cell at ith-row and j-th column of the cost matrix is:

costMatrix[i][j] = costOf(A[i-1], B[j-1]) + min(costMatrix[i -1][j], costMatrix[i, j-1], costMatrix[i - 1][j - 1])

The optimal value would located at the bottom-right cell of the cost matrix, and using that we can find the optimal path, the (i, j) position of the right bottom cell would be the first step in the optimal warping path. And then we scan the left, top and top-left cell of our current focusing cell, pick the one that has the least cost, the picked cell's (i, j) position would be our second step in the optimal warping path, and the picked cell would be our new focusing cell. Continue this process until we reach the cell (0, 0) which would be the last step in the optimal warping path.

By picking the least cost cell each time when moving to top-left of the matrix, we ensures that we end up with a path that has the minimised total cost.