Homework 1

Lucas Cosier

Data Mining I

ETH zürich

October 29, 2022

Pair of classes	Manhattan	DTW, $w = 0$	DTW, $w = 10$	DTW, $w = 25$	DTW, $w \to \infty$
abnormal:abnormal	67.77	67.77	38.65	26.48	25.37
abnormal:normal	67.52	67.52	34.20	26.94	26.35
normal:normal	45.65	45.65	24.42	22.17	21.87

Table 1: Reported DTW and Manhattan distances for abnormal and normal heartbeats.

Problem 1

a) **Solution.** Table 1 summarizes the statistics gotten by running the provided scripts with the necessary implementation required to solve task 1.a).

b) **Solution.** Per Table 1, the Manhattan distance between `abnormal:abnormal` and `abnormal:normal` is too small to distinguish between the two heartbeat groups. For DTW, since for $w = 0$ the comparisons are done only diagonally, this case reduces to the Manhattan distance, as seen in Table 1. Depending on which w is chosen, up to 3 groups can be distinguished. In practice, based on the preliminary data presented above, I would further run a grid search for w where the grid search is constrained in $[0, 10]$, as for $w \to \infty$ the intra and inter distances for abnormal and normal heartbeats become again not distinguishable. The optimal hyperparameter would be chosen as to maximize the absolute distance between all 3 groups.

c) **Solution.** The locality constraint w defines the time window that constrains the warping paths, similar to a sliding (time) window $[-w, +w]$. As discussed in the previous point, w-DTW can be reduced to Manhattan for $w = 0$, since the pairings are computed only on the diagonal. Since with $w \to \infty$ the set of possible warping paths increases, the noise injected by considering false pairings also increases with it, reason for which the separation *decreases*, as reflected in Table 1.

d) **Solution.** The DTW distance is not a metric. Counterexample:

Let $t_1 = [0, 0], t_2 = [0, 1], t_3 = [0, 1, 1]$.

Then, assume the triangle inequality holds, i.e.

$DTW(t_1, t_3) \leq DTW(t_1, t_2) + DTW(t_2, t_3)$

However, $DTW(t_1, t_3) = 2, DTW(t_1, t_2) = 1, DTW(t_2, t_3) = 0$

And since $2 \not\leq 1$ the property doesn't hold and hence the distance is not a metric.

e) **Solution.** For vanilla DTW the time complexity is $\mathcal{O}(n^2)$ assuming two time series of length n. This is due to considering all possible alignments to perform DTW. For constrained DTW, since the window has length $2w$, the time complexity decreases to $\mathcal{O}(n \times 2w) = \mathcal{O}(n \times w)$ and if we respect the big O convention, then the complexity can be written as $\mathcal{O}(n)$ if we consider w to be a constant.

Pair of classes	SP
mutagenic:mutagenic	5309.92
mutagenic:non-mutagenic	2706.77
non-mutagenic:non-mutagenic	1433.27

Table 2: Reported average SP kernel similarities between the abnormal and normal class.

Problem 2

b) **Solution.** The reported Shortest Path distances between the two groups are reported in Table 2.

c) (a) **Solution.** Assuming fixed matrix size n, the complexity of the Floyd Warshall algorithm is $\mathcal{O}(n^3)$. The implementation provided is already an improvement since we vectorize the inner-most loop to compute one column at a time. This of course can be parallelized by loop unrolling and blocking for cache size to compute multiple mini blocks for multiple columns at a time. A natural extension would be SIMD intrinsics. The runtime can further be increased by noting that the adjacency matrix doesn't need to be stored in doubles (double-precision floating-points). Depending on the size of the input, one can drastically reduce cache misses and working set size by reducing precision. In this case, we note that using unsigned 16-bit integers reduces the working set by 4× while keeping results sound. This is possible because we set the maximum cost to the maximum of an unsinged 8-bit integer (255) to avoid overflow problems.

(b) **Solution.** The total complexity of the SP kernel is $\mathcal{O}(n^4)$, assuming two matrices $S1, S2$ of size $n \times n, m \times m, m < n$. The SP kernel suffers from the following drawbacks:

- there are four loops and an if statement
- random memory access on $S1, S2$.

Ideally, one could try and hold the upper diagonal matrix that is looped over in the inner for loops in memory to avoid cache misses. If the size of the inner matrix is too big, one could block for locality and leverage techniques from MMM (matrix-matrix multiplication), since this computation is conceptually similar to MMM. Alternatively, one can convert the shortest path adjacency matrices using the sparse matrix representation to coordinate lists in order to reduce the working set. One would need to account for 1 array for the value, 1 array for the row, and 1 array for the column. Lastly, one can choose a kernel with better time complexity, like the subtree Weisfeiler-Lehman Kernel [1].

To be more pragmatic, for this homework, we note that using Numba's Just In Time compilation (JIT) to translate the Python code into pure machine code on the `sp_kernel` function yields ≈ 100× speed-up (measured on an Intel i7-1165G7 machine with 16 Gb of RAM, time measured for the groups reported in Table 2 was, in order, $\approx\ 1.32s, 0.33s, 0.08s$) since the function is written in pure Python and can be easily translated. The implementation with Numba can be tried out both for DTW and SP kernels by un-commenting the import and the decorators. Python is slow.

References

[1] Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper/2009/file/0a49e3c3a03ebde64f85c0bacd8a08e2-Paper.pdf.