Homework 2 – Data Mining I – John Oehninger

1b)

Pair of classes	Manhattan	DTW, w = 0	DTW, w = 10	DTW, w = 25	DTW, w = inf
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

Looking at my output data, it is immediately visible that the Manhattan and DTW (w=0) produce the same results. This is obvious, as with w=0, no flexibility is allowed, which in the end does nothing else but the same as the manhattan_distance function. It is important to note that there is no clear distinction between abnormal:abnormal and abnormal:normal. Therefore, these are not good options for use in practice.

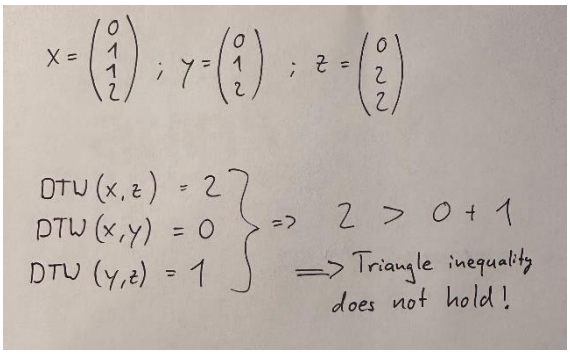
Looking at the far end of the output data, I can see that with w going towards infinity, the distances seem to converge to numbers within a range of 5. This is too close to clearly separate between groups, especially if compared to DTW (w=10), where I can see the clearest distinction between groups. DTW (w=10) is the one I would choose in practice.

1c) As mentioned above, as w goes towards infinity, it seems that all group distances converge to numbers within a range of approximately 5. I am thinking through what happens as w goes towards infinity. In this case, so many choices are generated, that the minimum distance of all the possibilities is almost always bound to be “small”. This seems logical to the reason of the converging distances. But this leads to the distance function actually describing randomness in the data and not the underlying relationship. Overfit models lead to poor prediction.

I feel like “occam’s razor” fits well in this description. We should not make the calculation too complex but stay in a reasonable feasible realm.

1d) These are the conditions that need to be fulfilled for a function to be a metric.

$$
\begin{aligned}
&1) d(x_1, x_2) \geq 0 \\
&2) d(x_1, x_2) = 0 \text{ iff } x_1 = x_2 \\
&3) d(x_1, x_2) = d(x_2, x_1) \\
&4) d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)
\end{aligned}
$$



Triangle inequality does not hold!

DTW is a distance measure but not a metric.

1e) Time complexity for DTW

If w is constrained, then depending on the constraint “level”, the time complexity is between O(n) (linear time) and O(n^2) (quadratic time) or O(n*m). If m = n = w (no constraints), the time complexity is of O(n^2), as every combination of n[i] to m[j] is considered, we have a nested loop, as taken from my code.

```
for i in range(1, n+1):
    for j in range(max(1, i-w), min(m, i+w) + 1):
        C[i,j] = abs(x[i-1] - y[j-1]) + min(C[i-1,j], C[i,j-1], C[i-1,j-1])
```

2c) With Floyd-Warshall we are moving in cubic time, O(n^3), since we have double nested loops.

```
for k in range(n):
    for i in range(n):
        for j in range(n):
            if A[i,j] > A[i,k] + A[k,j]:
                A[i,j] = A[i,k] + A[k,j]
```

In search of improvements, I stumbled upon Dijkstra’s algorithm which seems to work well for sparse graphs. This would result in O($VE+V^2logV$) time complexity, a Johnson’s algorithm. But for graphs that are connected throughout there doesn’t seem to be a possible improvement.

For the shortest path kernel, the time complexity is O(n^2) in how I implemented it in my code with a nested loop.

```
for i in range(n):
    for j in range(m):
        if upper1[i] == upper2[j]:
            K += 1
```

To improve the SP kernel time complexity, I stumbled upon an approach in a master’s thesis to achieve a shorter run time at the cost of error by using approximations.