Kyle Jackson
CS 674
Prof. Lin
2/22/2017

HW 1: Nearest Neighbor Classification using Euclidean Distance and Dynamic Time Warping

In this assignment, I implemented the One-Nearest Neighbor classification algorithm which can be generalized to K-Nearest Neighbor. My implementation uses Euclidean distance and dynamic time warping to determine similarity between two time series. As instructed, the accuracy of my results for (1) Euclidean distance, (2) DTW with no constraint, (3) DTW with warping window size of 20%, and (4) *k*-NN are shown in *Table 2*.

Nearest Neighbor Implementation

My implementation of the Nearest Neighbor classification algorithm was standard. The pseudo-code for the implementation is in *Table 1*. In accordance with (4) of the assignment, my extension to this problem is the use of *k*-NN. One thing to note about my implementation is that in the occasion that two or more labels are tied when $k > 1$, classification is based off of the first majority label encountered. For example, for list *my_list* = [1,2,3,4,3,4], although values 3 and 4 are tied for most frequent, because 3 appears first, it is selected as the label.

Another thing to point out about the implementation in *Table 1* is step 1. Each row in *d_matrix* corresponds to one test instance and each column corresponds to one train instance. Each cell contains the total distance between the test and train instance. To find the nearest neighbor, we simply sort each row, get the index of the smallest value or values, and then extract the labels from there.

Distance measures

Euclidean distance is calculated as follows for time series *q* and *p:*

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i + p_i)^2} \quad (1)$$

Table 1

Algorithm: k-Nearest Neighbor

```
Input:
    train, where time series $q_i \in$ train.txt and $i$ represents the file index
    test, where time series $c_j \in$ test.txt and $j$ represents the file index
    $k$, where $k \in \mathbb{Z}$ and indicates the number of nearest neighbors for classification
    distance metric, where $d \in$ {euclidean, DTW}
Output:
    predicted labels for all $c_i$

1. initialize N x n matrix d_matrix, where N and n represent number of time series in test and train files, respectively.
2. for $q_i$ in train:
3.     for $c_j$ in test:
4.         distance = d($q_i$, $c_{j,}$)
5.         d_matrix[$i$, $j$] = distance
6.
7. for $r$ in d_matrix:    # $r$ contains similarities between one $c_j$ and all $q_i$
8.     train_indices = get indices from $k$ smallest values in sort($r$)
9.     get labels from train_indices
10.    if $k$ == 1:
11.        classify $q_i$ using single label
12.    else:
13.        majority vote of labels
14.        if a tie exists:
15.            use first encountered frequent label
```

Dynamic time warping was implemented by following the class slides from lecture 2, [1], and [2]. My implementation calculates the entire distance matrix for two time series, as outlined in [1], using the squared distance between each point of the two time series. In the unconstrained window case, I then calculate the cumulative distance matrix across all points in the grid using equation (2) where q and c are two time series, q_i and c_j are two points on their respective time series, $d(q_i, c_j)$ is the distance in the current cell, and $\gamma(i,j)$ is the cumulative distance of $d(q_i, c_j)$ and the minimum of the cumulative distances from the three adjacent cells [1].

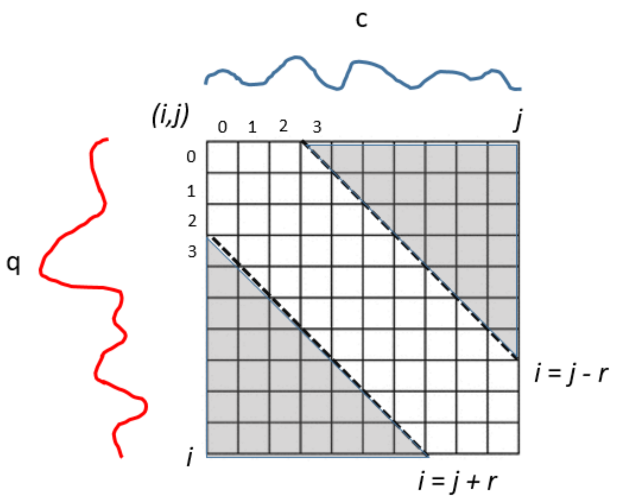
$$\gamma(i,j)=d(q_i,c_j)+min\{\gamma(i-1,j-1),\gamma(i-1,j),\gamma(i,j-1)\} \quad (2)$$

In the constrained case, the window width was calculated using the following:

$$r=wn \quad (3)$$

where w is the warping window constraint (e.g., 20%), n is the length of the time series, and r is the size of the actual window on one axis. Given that the slope $|m|$ of the window constraint is equivalent to 1 and an axis intercept of r, the constraints can be written as lines. Given axes i and j, the upper and lower window constraints can be written as $i = j - r$ and $i = j + r$, respectively.

Figure 1



In the example in *Figure 1*, *r* = 3. My implementation of the window constraints states that during the cumulative distance calculation for each cell, do not calculate the cumulative distance for the cell that is outside the window and in the "gray" zone.

The reason why the path in *Figure 1* goes from top left to bottom right, as opposed to bottom left to top right, was because of the way indexing numpy arrays in Python is done. Structuring the matrix such that the query time series *q* would be going from *i* to 0 as opposed to 0 to *i* in *Figure 1* would have required additional unnecessary preprocessing and would have made indexing difficult.

Results

The results were as expected (*Table 2*). 1-NN with Euclidean distance was relatively fast, only taking about 10 minutes to compute across all data sets. Another thing to note is that 1-NN performed better than *k*-NN, where *k* >1. Although *k*=10 is listed, I also tried *k*=5, which still underperformed 1-NN. This result is supported by previous work done showing that 1-NN is one of the top performing classifiers, as opposed to *k* > 1, as we have discussed in class.

More interesting were the results for Dynamic Time Warping. Across all datasets, 1-NN DTW without a warping window constraint performed the best in accuracy, but performed the worst in run time. As shown in *Table 2*, after running approximately 68 hours, no warping window DTW still did not complete the computation for Data set 5 alone. As of class today, it will have completed a little over half of the DTW calculations.

1-NN DTW with the warping window (3) completed all data sets in roughly 60 hours, with Data set 5 taking about 50 hours to complete. An unexpected result of this is how poorly it performed in terms of accuracy against the other methods.

Table 2: Results in accuracy

	Data set 1	Data set 2	Data set 3	Data set 4	Data set 5
(1) 1-NN Euclidean Dist	85.22%	78.29%	51.65%	78.88%	**99.55%**
(2) 1-NN DTW without w	**99.67%**	**83.43%**	**59.09%**	**79.20%**	null*
(3) 1-NN DTW with w (20%)	32.89%	82.29%	9.91%	21.76%	91.23%
(4) 10-NN, euclidean dist.	65.22%	71.43%	45.04%	67.36%	98.65%

* Data set 5 not finished running after approximately 68 hours

Discussion

The results from this experiment align with what has been shown to us in class and in the literature, except for 1-NN DTW with a window constraint of 20%. I did not anticipate it to perform so poorly. What is odd about the results is that it had high accuracy for data sets 2 and 5, but not the others. This raises the question as to whether there is an issue with my implementation. Future work on this would be to look at different values of *w* and potential flaws in my implementation.

Although I did not implement LB_Keogh or other variations of lower bounding, if I were to do this assignment again, I would. Implementing a vanilla version of DTW without *w* and with *w* made me appreciate how slow DTW is without speed ups. Further, my implementation still iterates through the cumulative distance matrix in *Figure 1* even if $\gamma(i,j)$ is outside the warping window constraint, but does not calculate gamma. To improve run time, I would alter my implementation such that iteration only occurs within the warping window.

For my Nearest Neighbor classifier implementation, future work involves making the handling of *k* >1 better. In particular, instead of the current implementation that takes the first most common value as the label in case of a tie, I would compute the sum of the distance measures for the same label, and then classify based on that.

References

[1] Ratanamahatana, C. A., & Keogh, E. (2004, August). Everything you know about dynamic time warping is wrong. In Third Workshop on Mining Temporal and Sequential Data.
[2] Ratanamahatana, C. A., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M., & Das, G. (2009). Mining time series data. In Data mining and knowledge discovery handbook (pp. 1049-1077). Springer US.