Dynamic Time Warping(DTW)

qhy

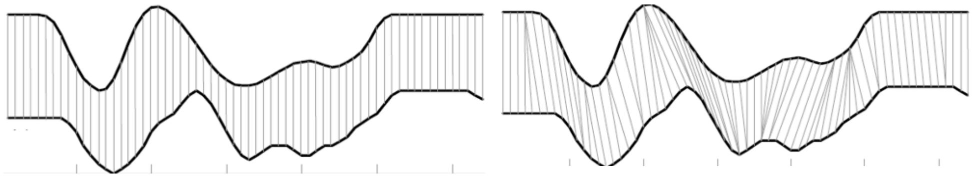
2018 年 1 月 13 日

目录

1 Dynamic Time Warping(DTW)

如何度量两个等长时间序列之间的距离?

基本的想法就是使用欧式距离,但是欧式距离有以下缺点: 欧式距离的局限在于其每个时间点对应的位置是唯一的, 也就是说,相同的两个时间序列,如果一个时间序列往后挪了小段时刻, 那么计算出来的欧式距离可能偏差很大。



(a) 欧式距离 (b) 其他

图 1: 两个时间序列距离的计算

对于左图,是普通的欧式距离,但如果采用右图的映射计算方式[1] ,计算得到值会比欧式距离小得多,也合理得多。

如何度量两个不等长的时间序列之间的距离?

想法是把不等长变得等长,简单地说,就是把短的序列变成和长序列等长的序列。

对于上述两个问题,可以使用**DTW**来解决。

首先是不等长的情况,从对应关系的角度来看,只需要短序列的一个点能对应长序列的多个点即可解决问题。

从对应关系的角度来看,序列的点之间的对应关系可能有多重,这里我们取距离最小的一个情况。

实际上,这是一个最短路径的动态规划问题。

那么对于两个序列Q,C,DTW的定义如下:

$$DTW(Q,C) = \min \sum_{k=1}^{K} w_k \tag{1}$$

其中,K表示路径的长路,w_k表示第k步对应的映射关系的平方和(即子序列$Q_{1\to i}$与子序列$C_{1\to j}$的欧式距离)。

要求:$w_1 = d(1,1), w_K = d(n,m)$

具体推导公式为:

$$g(i,j) = \min \begin{cases} g(i-1,j) + d(i,j) \\ g(i-1,j-1) + d(i,j) \\ g(i.j-1) + d(i,j) \end{cases} \tag{2}$$

[1]即一个点映射多个点,相应的,另一边的一个点也要映射多个点回去才能平衡

2 另一种DTW

那么对于两个序列Q,C,DTW的定义如下:

$$DTW(Q,C) = \min \frac{1}{K}\sqrt{\sum_{k=1}^{K} w_k} \tag{3}$$

这么定义是为了在上面的DTW中,DTW更倾向于走对角线的路径,因为路径步骤更短, 因此,这里进行了归一化处理。
这个表达式不能直接用动态规划求解,这里采用一个近似算法:

$$DTW(Q,C) = \min \frac{1}{K}\sqrt{\sum_{k=1}^{K} w_k} \approx \min \frac{1}{n+m-1}\sqrt{g(n,m)} \tag{4}$$

则有以下动态规划过程:

$$g(i,j) = \min \begin{cases} g(i-1,j) + d(i,j) \\ g(i-1,j-1) + 2d(i,j) \\ g(i.j-1) + d(i,j) \end{cases} \tag{5}$$

这里对走对角线的操作给与双倍的惩罚($\times 2$),就使得$K = m+n-1$

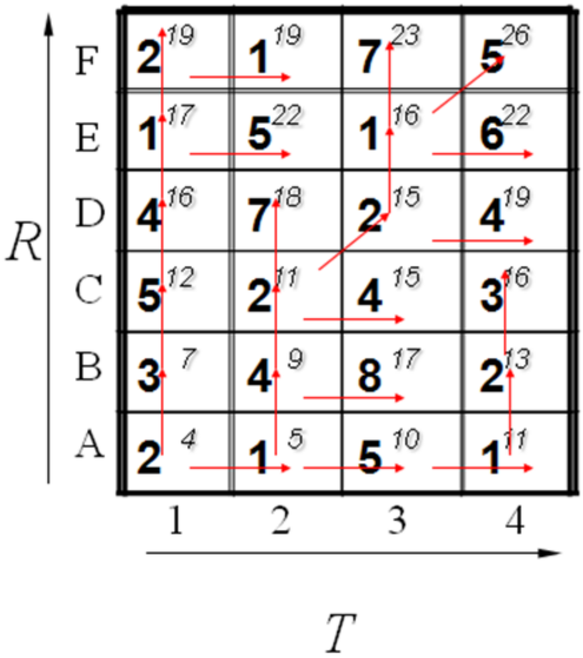


图 2: DTW的动态规划过程:初始是从(0,0)开始走,因此(1,1)是$2 \times 2 = 4$

3 Piecewise Dynamic Time Warping(PDTW)

Piecewise Aggregate Approximation(PAA):
每段间隔的数据取均值作为这一段的代表以进行数据的压缩,从而降低计算的开销。

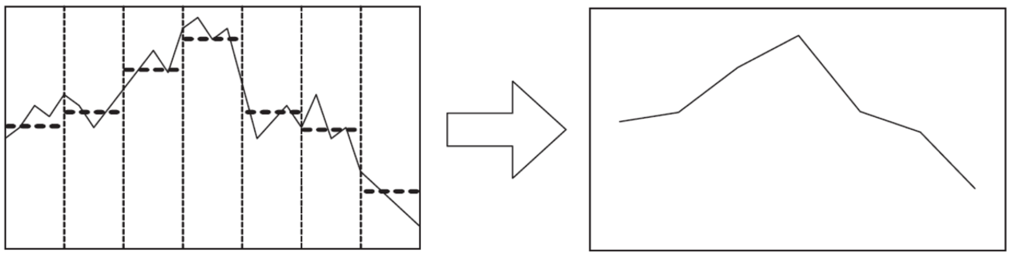


图 3: Time series dimensionlity reduction by PAA.The horizontal dotted lines show the mean of each segment.

PDTW = PAA + DTW

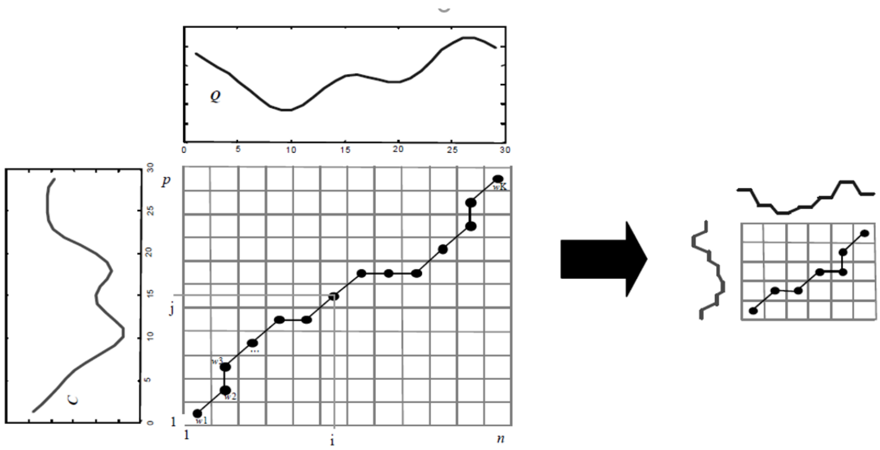


图 4: Example of warping path with DTW and PDTW

注意:$PDTW$ 虽然减少的计算的开销,但是精度却大大降低了。

4 Iterative Deepening DTW(IDDTW 迭代加深DTW)

$PDTW$虽然减少了计算的开销,但是却大大减少了精度。
这里的$IDDTW$虽然还是以DTW为名,但更准确地说,应该是在$k-NN$问题中, 求k近邻的时候,使用迭代加深的思想,提前淘汰不满足要求(距离很大)的时间序列, 从而避免计算真实DTW以减少计算开销。
$IDDTW$在保持原来计算精度的情况下,使用迭代加深的思想利用$PDTW$ 对一定不满足条件的序列进行提前淘汰,而不用计算到完全的DTW, 从而在保证精度的同时,降低计算的开销。
注:计算的开销和数据的排列有关,最严重的情况,时间开销会比正常的DTW还要多$\frac{1}{3}$

如何判断一个序列应该被提前淘汰?

```
Algorithm BuildErrorDistribution(N,L)
// N = dataset, L = number of depths
// d = {1,..,L}, C = {d range of compression rates}
for sample size i to build error distributions
  {Randomly pick 2 sequences of same lengths, N_a and N_b, where a != b}
  R_i := DTW(N_a, N_b);                    // True DTW of N_a and N_b
  for the range of d
    Approx_ N_a := PAA(N_a, C_d);          // Dimensionality reduced representation of N_a and N_b
    Approx_ N_b := PAA(N_b, C_d);
    RE_id := DTW(Approx_ N_a, Approx_ N_b);      // PDTW_d of N_a and N_b
    E_id :=RE_id – R_i;                          // Error between DTW and PDTW_d
    P_d :={P_d, E_id };                          // Accumulates each error at d in P_d
  end for;
  StdDev_d is found from P_d               // StdDev_d is the standard deviation for each P_d
end for;
```

Table 2: Algorithm to build the error distributions.

在深度为L下,我们并不能判断真实的DTW是否比当前最好的DTW更小, 这里通过采样的方式,计算在深度L的情况下, 误差的分布(这个分布,用来估计真实的DTW与当前深度下的$PDTW$的差距)。

那么怎么使用这个分布?

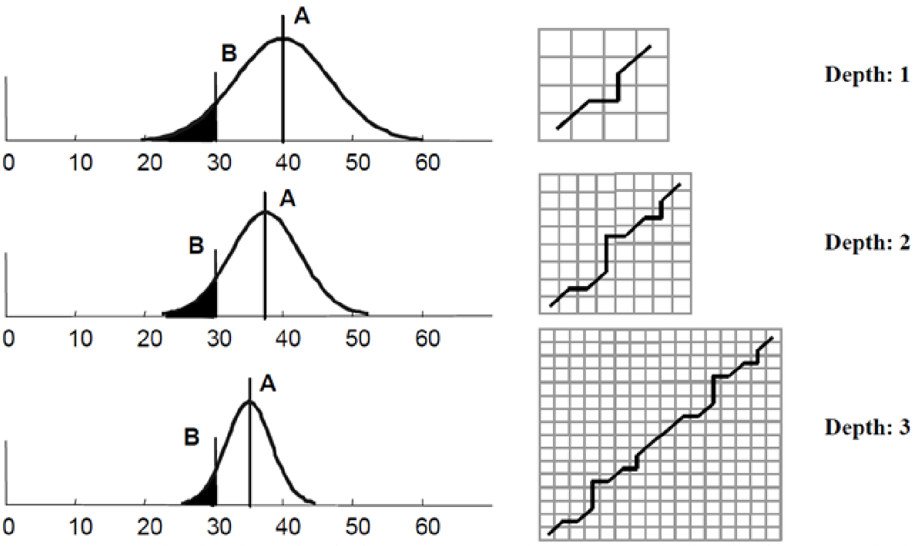


Figure 8: Demonstrates the intuition behind IDDTW. We have an error distribution at each depth. **A** is the approximated distance and **B** is the best_so_far. The mean of the distribution is at 0. If we center the distribution around **A**, then the distance between **A** and **B** is the approximated distance error. The shaded area can be seen as the probability that **A** could be better than the best_so_far. This also demonstrates the complexity of finding a warping path at each level of approximation.

在深度为1的情况下,我们有如图所示的一个分布,A表示当前深度下的DTW, 而B表示当前最小的DTW ,这样,黑色的面积就表示当前时间序列,真实的DTW 小于B的可能性。
当这个可能性小于某个阈值的时候,我们就淘汰掉这个时间序列。(如果一个时间序列始终不被淘汰,那么它就会不断加深, 直到计算出真实的DTW)
上述算法的分布,应该在$k-NN$算法之前预处理出来,而不是$k-NN$过程中计算。