武汉大学

课程名称：并行算法

学　　院	计算机学院
专业名称	软件工程
学　　号	2019202110095
学生姓名	蒋文君
任课教师	何国良

2019 年 **12** 月 **29** 日

目　录

基于 DTW 相似度的 AP 聚类的并行优化

摘　　要：DTW 是一种衡量两个时间序列的相似度的方法。因为其高效地解决了数据不对齐的问题，在计算两个不等长序列之间的相似度上被广泛使用。然而它的时间成本很高，尤其是在高维数据上。本文围绕 AP 聚类，基于 DTW 计算其样本间的相似度。为了提高基于 DTW 相似度计算的速度，从改进算法本身和并行处理两个角度进行了优化，提出了三种方法：fastdtw、fastdtw_parallel、tf_dtw，通过理论分析和实验测试，最终证明基于并行实现的 fastdtw_parallel 和 tf_dtw 速度上得到了很高的提升，远优于 fastdtw。

关键词：DTW，AP 聚类，并行优化

1 引言

基于处理器制造工艺的提升接近极限，单纯靠提高主频来提升性能已不再适应时代需求，促使处理器从单核向多核转化。经过近年发展，多核处理器在当前已经成为主流配置，但是目前大部分编写的程序还是串行的，极大地浪费了处理器的计算资源。另一方面，随着科学技术的发展，所获得的信息数据越来越庞大，构建一个高效的并行化的算法，必将是未来发展的一大趋势。

DTW 是一种衡量两个时间序列的相似度的方法。尤其适用于不同长度、不同节奏的时间序列。DTW 将自动 warping 扭曲时间序列，即在时间轴上进行局部的缩放，使得两个序列的形态尽可能的一致，得到最大可能的相似度。大部分时间序列挖掘算法都使用基于距离的搜素作为核心的方法，用于处理不等长度的时间序列主要是使用 DTW 算法，而用这种算法在较大数量级的时间序列上，比如 10 亿级别的数据，搜索时间过长一直是瓶颈。

因此，本文主要基于 DTW 相似度的 AP 聚类算法。文章整体结构如下：第 2 部分介绍基于 DTW 相似度的 AP 聚类算法的相关知识，主要包括 AP 算法和 DTW 算法；第 3 部分从改进算法本身和并行处理两个角度对基于 DTW 相似度的 AP 聚类算法进行优化；第 4 部分实验对比三种优化方法之间的速度；第 5 部分对整篇文章进行总结。

2 背景知识

2.1 AP 聚类

AP(Affinity Propagation)[1]通常被翻译为近邻传播算法或者亲和力传播算法，是在 2007 年的 Science 杂志上提出的一种新的聚类算法。AP 算法的基本思想是将全部数据点都当作潜在的聚类中心(称之为 exemplar)，然后数据点两两之间连线构成一个网络(相似度矩阵)，再通过网络中各条边的消息(responsibility 和 availability)传递计算出各样本的聚类中心。

算法描述：

设$\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_n\}$数据样本集，数据间没有内在结构的假设。令 $\boldsymbol{S}$ 是一个刻画样本点之间相似度的矩阵，使得 $\boldsymbol{s(i,j)>s(i,k)}$当且仅当$\boldsymbol{x_i}$与$\boldsymbol{x_j}$的相似性程度要大于其与$\boldsymbol{x_k}$的相似性。

AP 算法定义了两个信息矩阵：吸引信息矩阵 R 和归属信息矩阵 A，基于相似矩阵 S 对吸引信息矩阵 R 和归属信息矩阵 A 不断迭代更新。其中，$\boldsymbol{R[i,k]}$描述了子空间$\boldsymbol{W_k}$适合作为子空间$\boldsymbol{W_i}$的聚类中心的程度，表示的是从$\boldsymbol{W_i}$到$\boldsymbol{W_k}$的消息；$\boldsymbol{A[i,k]}$描述了子空间$\boldsymbol{W_i}$选择子空间$\boldsymbol{W_k}$作为其据聚类中心的适合程度，表示从$\boldsymbol{W_k}$到$\boldsymbol{W_i}$的消息。

两个矩阵 R ,A 初始全部为 0，根据下述公式进行迭代更新：

首先，吸引信息$\boldsymbol{R_{i+1}^{[i,k]}}$按照下式进行迭代更新。

$$\boldsymbol{R_{t+1}[i,k]} = \chi\boldsymbol{[i,k]} - \boldsymbol{max_{k^{'}\neq k}\{A_t[i,k^{'}]} + \chi\boldsymbol{[i,k^{'}]\}}$$

然后，归属信息$\boldsymbol{A_{t+1}[i,k^{'}]}$按照下面的式子迭代:

$$\boldsymbol{A_{t+1}[i,k] = min\left(0,R_t[k,k] + \sum_{i^{'}\notin\{i,k\}} max\{0,R_t[i^{'},k]\}\right),i\neq k}$$

$$\boldsymbol{A_{t+1}[k,k] = \sum_{i^{'}\neq k} max\{0,R_t[i^{'},k]\}}$$

对以上步骤进行迭代，如果这些决策经过若干次迭代之后保持不变或者算法执行超过设定的迭代次数，则算法结束。将最终得到的矩阵 R，A 相加，遍历相加后的矩阵的每一行，该行样本属于每一行最大值所在的索引。

2.2 DTW 距离

Dynamic Time Warping(dtw)[2]是一种衡量两个长度不同的时间序列的相似

度的方法。因为其高效地解决了数据不对齐的问题，在计算两个不等长序列之间的相似度上被广泛使用。

设有两个长度不相同的序列 $\boldsymbol{X}=\{\boldsymbol{x_1},\boldsymbol{x_2},\ldots,\boldsymbol{x_n}\}$, $\boldsymbol{Y}=\{\boldsymbol{y_1},\boldsymbol{y_2},\ldots,\boldsymbol{y_m}\}$，DTW 的计算过程如下：

为了对齐两个序列，需要构造一个 $\boldsymbol{n\ x\ m}$ 的矩阵网格，矩阵元素$(\boldsymbol{i},\boldsymbol{j})$表示$\boldsymbol{x_i}$和$\boldsymbol{y_j}$两个点的距离 $\boldsymbol{d}(\boldsymbol{x_i},\boldsymbol{y_j})$。也就是序列 $\boldsymbol{X}$ 的每一个点和 $\boldsymbol{Y}$ 的每一个点之间的相似度，距离越小则相似度越高。一般采用欧式距离，即 $\boldsymbol{d}(\boldsymbol{x_i},\boldsymbol{y_j})=(\boldsymbol{x_i}-\boldsymbol{y_j})^{\boldsymbol{2}}$, 每一个矩阵元素$(\boldsymbol{i},\boldsymbol{j})$表示点$\boldsymbol{x_i}$和$\boldsymbol{y_j}$的对齐程度。DTW 算法可以归结为寻找一条通过此网格中若干格点的路径，路径通过的格点即为两个序列进行计算的对齐的点。

这条路径被定义为 warping path 规整路径，并用 W 来表示， W 的第 k 个元素定义为$\boldsymbol{w_k}=(\boldsymbol{i},\boldsymbol{j})_{\boldsymbol{k}}$，定义了序列 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 的映射，则 $\boldsymbol{W}=\{\boldsymbol{w_1},\ldots,\boldsymbol{w_k},\ldots,\boldsymbol{w_p}\}$。这条路径不是随意选择的，需要满足以下几个约束：

1) 边界条件：$\boldsymbol{w_1}=(\boldsymbol{1},\boldsymbol{1})$和$\boldsymbol{w_p}=(\boldsymbol{m},\boldsymbol{n})$。任何一个序列其各部分的先后次序不可能改变，因此所选的路径必定是从左下角出发，在右上角结束。
2) 连续性：如果$\boldsymbol{w_{k-1}}=(\boldsymbol{a'},\boldsymbol{b'})$，那么对于路径的下一个点$\boldsymbol{w_k}=(\boldsymbol{a},\boldsymbol{b})$需要满足 $(\boldsymbol{a}-\boldsymbol{a'})<=\boldsymbol{1}$ 和 $(\boldsymbol{b}-\boldsymbol{b'})<=\boldsymbol{1}$。也就是不可能跨过某个点去匹配，只能和自己相邻的点对齐。这样可以保证 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 中的每个坐标都在 W 中出现。
3) 单调性：如果$\boldsymbol{w_{k-1}}=(\boldsymbol{a'},\boldsymbol{b'})$，那么对于路径的下一个点$\boldsymbol{w_k}=(\boldsymbol{a},\boldsymbol{b})$需要满足 $\boldsymbol{0}<=(\boldsymbol{a}-\boldsymbol{a'})$和 $\boldsymbol{0}<=(\boldsymbol{b}-\boldsymbol{b'})$。这限制 W 上面的点必须是随着时间单调进行的。

结合连续性和单调性约束，每一个格点的路径就只有三个方向了。对于网格矩阵的每个元素 $\boldsymbol{D}(\boldsymbol{i},\boldsymbol{j})$，有下述定义：$\boldsymbol{D}(\boldsymbol{i},\boldsymbol{j})=\boldsymbol{d}(\boldsymbol{x_i},\boldsymbol{y_j})+\min\{\boldsymbol{D}(\boldsymbol{i}-\boldsymbol{1},\boldsymbol{j}),\mathrm{D}(\boldsymbol{i},\boldsymbol{j}-\boldsymbol{1}),\boldsymbol{D}(\boldsymbol{i}-\boldsymbol{1},\boldsymbol{j}-\boldsymbol{1})$。

使用动态规划策略，最终寻找到一条通过此网格中若干格点的最短路径：

$$\boldsymbol{DTW}(\boldsymbol{X},\boldsymbol{Y})=\sqrt{\boldsymbol{min}\left\{\sum_{\boldsymbol{k}=\boldsymbol{1}}^{\boldsymbol{p}}\boldsymbol{d}(\boldsymbol{w_k})\right\}}$$

3 基于 DTW 相似度的 AP 聚类优化

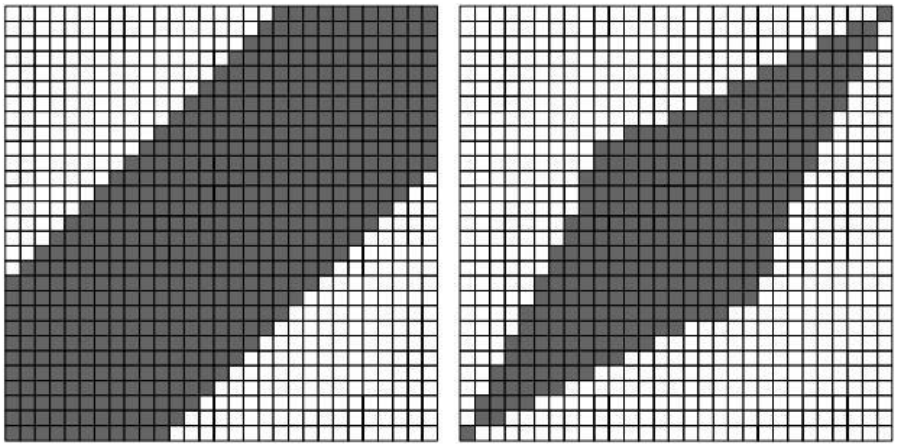
根据 AP 聚类的算法描述，易知基于 DTW 相似度的 AP 聚类耗费的时间主要是

在相似矩阵的计算上。为了提高基于 DTW 相似度计算的速度，有以下几种改进方法。

3.1 fast-DTW

一般的 DTW 实现时间复杂度是 $\boldsymbol{O(T^2)}$。Stan Salvador 在 2007 年提出了 fast-DTW，能达到近似线性 $\boldsymbol{O(T)}$的时间计算 DTW。

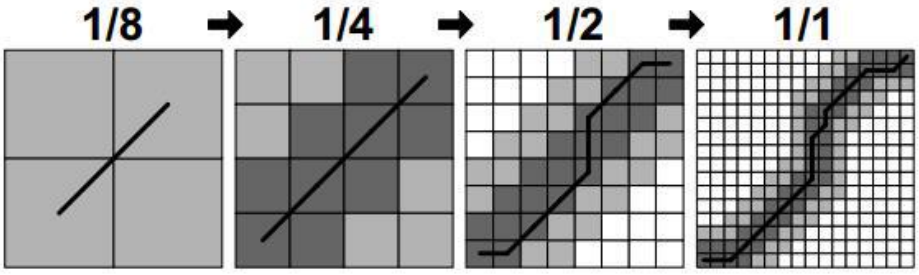
fast-DTW[3]是基于 DTW 的改进，该算法最主要有两个部分。第一个是约束。亦即减少网格矩阵 D 的搜索空间，下图中阴影部分为实际的探索空间，空白的部分不进行探索，减少了搜索的次数。第二个是抽象，将图像的像素合并，$\boldsymbol{1/1->1/2->1/4->1/8}$…，直到可以确定路径。



具体过程主要分为三步：

1) 粗粒度化。亦即首先对原始的时间序列进行数据抽象，数据抽象可以迭代执行多次 $\boldsymbol{1/1->1/2->1/4->1/16}$，粗粒度数据点是其对应的多个细粒度数据点的平均值。
2) 投影。在较粗粒度上对时间序列运行 DTW 算法。
3) 细粒度化。将在较粗粒度上得到的归整路径经过的方格进一步细粒度化到较细粒度的时间序列上。除了进行细粒度化之外，我们还额外的在较细粒度的空间内额外向外(横向，竖向，斜向)扩展 K 个粒度，K 为半径参数，一般取为 1 或者 2.

fast-DTW 算法的具体执行流程如下图所示：



第一个图表示在较粗粒度空间$(\mathbf{1/8})$内执行 DTW 算法。第二个图表示将较粗粒度空间$(\mathbf{1/8})$内求得的归整路径经过的方格细粒度化，并且向外(横向，竖向，斜向)扩展一个(由半径参数确定)细粒度单位后，再执行 DTW 得到的归整路径。第三个图和第四个图也是这样。

由于采取了减少搜索空间的策略，fast-DTW 并不一定能够求得准确的 DTW 距离，但是 fast-DTW 算法的时间复杂度比较低，为 O(N)。

3.2 并行优化

3.2.1 基于 CPU

因为 DTW 的运算在样本维度上是互不依赖的，所以可以做并行。对于 $\boldsymbol{N}$ 个样本，我们需要计算的距离矩阵大小为 $\boldsymbol{N \times N}$，将其划分为 $\boldsymbol{k \times k}$ 块，可以同时进行运算，将各子矩阵拼接起来即可。同时，由于距离矩阵是个对称阵，因此只需要计算上三角或下三角即可，即并行数为 $\boldsymbol{k(k+1)/2}$，在 CPU 核心数恒定的情况下，这个技巧能够使 k 增大，从而节省时间。

3.2.2 基于 GPU

虽然 fast-DTW 在众多 DTW 实现中已经很快了，但是当样本量很多的时候，还是比较慢。从样本维度上并行，可用 GPU 加速，并用 tensorflow 实现，同时运行几个大小为 B 的 Mini-Batch。

3.3 时间消耗理论对比分析

计算耗时总是与样本数的平方成正比的，CPU 版与 GPU 版并行的区别在于，制约 CPU 并行运算上限是 CPU 核数，同时运算的子矩阵数不能超过 CPU 核数，而制约 tensorflow 版运算上限的是显存，可以通过设置较大的 Batch_Size，同时计算大量样本之间的距离，而且由于样本之间相互独立，所以每个 Batch 的运算时间实际上只与样本长度有关，当样本长度固定，每个 Batch 的运算时间也固定。

计算一对样本的 DTW 距离，设单核串行运算耗时$\boldsymbol{t_1}$秒，15 核并行耗时$\boldsymbol{t_2 = t_1/k}$（k 不一定能达到 15），tensorflow 版跑一个大小为 B 的 Mini-Batch 耗时$\boldsymbol{t_3}$秒。现有 $\boldsymbol{N}$ 个样本，需要获得 $\boldsymbol{N \times N}$ 距离矩阵。三种方法的时间消耗理论上如下所示，$\boldsymbol{T_1}$, $\boldsymbol{T_2}$, $\boldsymbol{T_3}$分别代表 fast-DTW，CPU 并行与 GPU 运行。

$$\boldsymbol{T_1 = t_1 \times \frac{N(N-1)}{2}}$$

$$T_2 = \frac{t_1}{k} \times \frac{N(N-1)}{2}$$

$$T_3 = t_3 \times \frac{N(N-1)}{2B} = t_3 \times Batch\_Num$$

由此可见，随着样本数增多，单核串行运算耗时呈指数增长，多核并行只能在前者的耗时上乘以一个小数，稍微打点折扣，而 GPU 版本在样本数不是很多时，只要不超显存，设置一个足够大的 Batch_Size 能有效减少所需 Batch 个数，从而大幅节省时间，而当样本数足够多时，耗时将与 Batch 个数呈正比，而 Batch 个数是与样本数的平方N^2成正比的，因此样本数很多时，GPU 版的耗时也将乘指数增长，但相比于 CPU 运算，仍有明显优势。

4 实验

数据集选用多元时序序列数据集 CharacterTrajectories，包含 20 种小写英文字母手写体的坐标序列。在计算多元时间序列之间的 DTW 距离时，将多个变量之间的 DTW 简单相加作为整个多元时间序列之间的 DTW 距离。

下表为数据集的具体信息：

训练集	测试集	类别数	序列长度	变量个数	数据类型
1422	1436	20	182	3	Motion Gesture

以下是 fast-DTW 和 tensorflow 版本 DTW 矩阵计算的时间消耗对比。tf_dtw 根据样本量的不同需要设置不同的 batch_size，过大的 batch_size 有使 GPU 显存不足的风险。

N	N(N-1)/2	fastdtw	fastdtw_parallel(cpu=15)	tf_dtw(batches)
10	45	8.07s	1.29s	16.31s (1)
50	1,225	229.88	26.60	72.30 (5)
100	4,950	959.57	102.74	72.97 (5)
200	19,900	~4k (1.1h)	402.69	77.13 (5)
1,000	499,500	~100k(27h)	~10k (2.8h)	795.20 (50)
2,858	4,082,653	~784k (9d)	~78.4k (21h)	6476.16 (400)

当样本数上升到 1000，单核 CPU 运算已经至少要 1 天时间才能算完了，对整个数据集的 2858 样本进行运算，估计需要 9 天时间，15 核并行运算可以缩减为 21 小时，而 GPU 版凭借较大的 Batch_size（10000），可以将时间缩减为 1.8 小时。

得到训练集样本两两之间的 DTW 矩阵后，将这个距离矩阵的负值作为相似度

矩阵（距离越大，负值越小，相似度越小），作为AP聚类的输入，获得若干个“中心”（centers），这样整个基于DTW相似度的AP聚类模型就训练完了。

测试阶段，从DTW距离矩阵上读出这些“中心”（centers）对应的行、测试样本对应的列，距离最小的center即为距离该测试样本最近的“中心”，将该中心的类别作为测试样本的预测类别，这样预测就完成了。

最终得到训练集的识别准确率为86.85%，测试集为83.43%。

5 结论

本文主要基于利用DTW计算相似距离的AP聚类展开，提高聚类效率。根据AP聚类算法的描述，AP聚类耗费的时间主要体现在利用DTW计算相似矩阵的计算上。为了加速基于DTW相似度计算的速度，分别提出三种改进方法：fastdtw、fastdtw_parallel、tf_dtw，通过理论分析和实验测试，最终证明基于并行实现的fastdtw_parallel和tf_dtw要远优于fastdtw，基于GPU并行的tf_dtw的速度要优于基于CPU的fastdtw_parallel。

6 参考文献

[1] B.J.Frey and D.Dueck,”Clustering by passing messages between data points”,Science,vol.315,pp.972-976,2007

[2] Shokoohi-Yekta M,Hu B,Jin H,et al. Generalizing DTW to the multi-dimensional case requires an adaptive approach[J]. Data Mining and Knowledge Discovery, 2017, 31(1):1-31.

[3] Stan Salvador, and Philip Chan.“FastDTW: Toward accurate dynamic time warping in linear time and space.” Intelligent Data Analysis 11.5 (2007): 561-580.