计算机研究与发展 DOI:10.7544/issn1000-1239.2019.20170741
Journal of Computer Research and Development 56(3):594-610,2019

基于相似性连接的时间序列 Shapelets 提取

张振国[1,2] 王 超[2] 温延龙[2] 袁晓洁[3]
[1](延边大学计算机科学与技术系 吉林延吉 133002)
[2](南开大学计算机学院 天津 300350)
[3](南开大学网络空间安全学院 天津 300350)
(zhangzhenguo@dbis.nankai.edu.cn)

Time Series Shapelets Extraction via Similarity Join

Zhang Zhenguo[1,2], Wang Chao[2], Wen Yanlong[2], and Yuan Xiaojie[3]
[1](*Department of Computer Science and Technology, Yanbian University, Yanji, Jilin* 133002)
[2](*College of Computer Science, Nankai University, Tianjin* 300350)
[3](*College of Cyber Science, Nankai University, Tianjin* 300350)

Abstract For time series classification, the classifier built by Shapelets has high classification accuracy and, meanwhile, the classification results are easily interpretable. Therefore, the extraction of discriminative Shapelets has attracted a lot of attention in the field of time series data mining. Research on Shapelets extraction has obtained promising achievement, but there are still some problems. The main reason is that the traversal of all time series subsequences to find the discriminative Shapelets is extraordinarily time consuming. Although some pruning techniques can be applied to accelerate the extraction process, they usually reduce the classification accuracy. In this paper, we propose a novel Shapelets extraction method based on similarity join, which abandons the idea of computing each subsequence's discriminative power. In the proposed method, each subsequence is considered as a basic computing unit and the similarity vector of two time series is obtained by the similarity join calculation of their subsequences. For the time series with different class label, we compute the difference vector of each time series pair and merge them into a candidate matrix which represents the differences between different time series class. Thus, we can easily obtain the eligible Shapelets from the candidate matrix. Extensive experimental results in real time series datasets show that, compared with the exist Shapelets extraction methods, the proposed method has high time efficiency while ensuring excellent classification accuracy.

Key words time series; Shapelets; similarity join; difference vector; candidate matrix

摘 要 在时间序列分类问题中,以 Shapelets 特征为基础的分类算法具有很高的分类准确率和良好的可解释性,因此,高辨别能力 Shapelets 的提取已成为时间序列研究领域重要的研究热点之一.对于 Shapelets 提取的研究已取得了很多优秀的成果,但仍存在一些问题,主要是由于通过遍历所有子序列来获取 Shapelets 的方式非常耗时.尽管可以采取剪枝策略优化该过程,但往往会损失分类准确率.为

收稿日期:2017-09-29;**修回日期**:2018-08-27
基金项目:国家自然科学基金项目(61772289);吉林省教育厅"十三五"科学技术项目(JJKH20191125KJ)
This work was supported by the National Natural Science Foundation of China (61772289) and the "13th Five-year" Science and Technology Project of the Jilin Provincial Education Department (JJKH20191125KJ).
通信作者:袁晓洁(yuanxj@nankai.edu.cn)

此,提出一种基于相似性连接的 Shapelets 提取方法,该方法舍弃逐一判断子序列分类能力的策略,而是以子序列为单位,通过相似性连接的思想构建时序数据间的相似性向量.对于不同类别的时序数据,计算每一对时序数据间的差异向量,进而得到表示时序数据集中不同类别间差异的候选矩阵,然后根据候选矩阵的数值差异,快速筛选出具有高分类能力的 Shapelets 集合.在真实数据集上的大量实验表明:相比于现有的 Shapelets 提取方法,这种相似性连接方法所得到的 Shapelets 在分类任务中不仅具有很好的时间效率,而且能保证高分类准确率.

关键词 时间序列;Shapelets;相似性连接;差异向量;候选矩阵

中图法分类号 TP18; TP391.4

近年来,对于时间序列数据(time series,简称时序数据)的研究与分析受到越来越多的关注,已成为数据挖掘领域的一个重要的研究课题[1-2],其原因在于时序数据广泛地存在于我们日常生活的诸多领域,如医疗、金融、运动轨迹以及天文星系观测等[3].一般来说,时序数据是由某一观测量随时间变化产生的,每一数据值之间有严格的时间先后顺序,这是时序数据区别于其他类型数据的本质特征.广义上,任何具有严格先后顺序的数值构成的序列,都可以称为时序数据.作为一项基础工作,时序数据的分类是被研究最多的问题之一,具有重要的应用价值,如在心电图信号分析中,对其进行分类能够快速鉴别信号的正常与异常,有助于辅助医疗;在工业生产中,对设备性能数据的采集与分类,能够及时了解设备的运行状况.相比于其他数据的分类,在提取时序数据特征,构建分类器的过程中,数据之间的先后顺序是必须要考虑的关键因素.因此,以时序数据整体作为处理对象,利用数据间相似性的最近邻分类器(nearest neighbor classifier, NN)被首先用于时序数据的分类问题[4-5].

基于 NN 分类器的时序分类算法的核心在于如何计算时序数据间的相似性.由于噪音、采集时间上的对齐等因素,时序数据在获取过程中可能会存在一些轻微的幅值偏移,这就使得在计算时序数据间的相似性时,相似性算法必须能够处理这种时间轴上的数据错位现象.有 2 种类型的相似性计算方式被广泛应用:1)基于动态时间规整(dynamic time warping, DTW)的距离计算[6];2)基于编辑距离(edit distance)的相似度计算[7].DTW 及其相关改进算法,如导数 DTW(derivative DTW)[8]、权重 DTW(weighted DTW)[9],通过寻找一条有效的规整路径来最小化对应的时序数据点间的总距离并借此计算 2 条时序数据的相似性.DTW 已被认为是时序数据相似性计算的基准[5].同 DTW 一样,编辑距离也通过寻找数据间的最佳匹配来计算相似度,所不同的是,基于编辑距离的方法,如时间规整编辑(time warp edit, TWE)[10]、移动-分裂-融合(move-split-merge, MSM)[11],允许数据间存在跨越和改变.这 2 种类型的时序数据相似性计算被称为弹性距离度量(elastic distance measures).在很长一段时间内,使用这些弹性距离度量的简单 NN 算法在时序数据分类问题上占据主要地位且很难被超过[12].然而,由于 NN 算法需要存储和搜索整个时序数据集,而这些弹性距离度量的计算又比较耗时(通常情况下,时间复杂度为 $O(m^2)$,m 为时序数据长度),这就导致在处理较大时序数据集时,时间和空间的消耗都比较大,限制了其应用.另一方面,这种方法是将时序数据作为整体来考虑的,然而更多的时候,对于时序数据类别区分起关键作用的是局部信息而非全局信息,同时,局部信息更能体现出时序数据间的本质差异.

为了解决上述 2 个问题,Ye 等人[13]提出使用子序列(subsequence)作为区分不同类别的依据,并将具有高类别可区分的子序列称为 Shapelets.相比于使用 NN 分类器的算法,Shapelets 提供了良好的可解释性,易于体现不同类别时序数据之间的关键差异.因此,Shapelets 一经提出便受到研究者的关注.如何快速提取高质量的 Shapelets 已成为最近几年时序数据处理领域的一个热点问题.然而,由于任意长度的时序子序列都可能是 Shapelets,而子序列数量庞大(例如长度为 60,时序数量为 600 的数据集,子序列数量达 1.098×10^6),所以逐一判断子序列的类别可分能力是一项非常耗时的工作.针对该问题,Ye 等人[13]提出使用剪枝策略加速子序列分类能力的判断.这些策略对于小数据集是可行的,但对于大数据集仍难以处理.为了提高 Shapelets 提取效率,许多使用加速技术的算法被提出,如使用空间换时间的 Logical Shapelets 算法[14]、借助时序数据

的离散化 SAX(symbolic aggregate approximation)[15] 表示的 Fast Shapelets 算法[16] 等. 这些方法虽然能够在一定程度上加速 Shapelets 的提取,但仍需要大的时间消耗,且会损失一定的预测准确率. 除此之外,Hills 等人[17] 更改子序列分类能力度量方法,使用易于计算的 Kruskal-Wallis 度量和 F-statistic 作为衡量标准,但加速效果并不明显,并且降低了分类准确率.

除了上述直接使用提取的 Shapelets 构建分类器外,Lines 等人[18] 提出了一种使用 Shapelets 构建对应时序数据特征向量的方法,可以使用当前比较成熟的分类器,如 SVM,而不必拘束于 Ye 等人[13] 使用的决策树分类器. 使用时序数据的特征向量表示,一方面可以保证 Shapelets 的可解释特性,另一方面可以充分利用成熟分类器的优点,提高分类准确率. 该方法的关键问题仍然是如何提取出高质量的 Shapelets,然而作者并没有改变辨别子序列分类能力的方式,故其方法仍然非常耗时.

为此,本文在 Yeh 等人[19] 所提出的时间序列 matrix profile 基础上,提出了一种基于相似性连接的时间序列数据 Shapelets 提取方法. 该方法舍弃了现有工作中先生成子序列,再对子序列进行区分能力判断的方案,而是通过差异向量体现不同类别时序数据本身的差异,利用相似性连接策略快速计算出隐含高分类能力 Shapelets 的候选矩阵,然后通过对比候选矩阵中的数值差异,筛选出 Shapelets 集合. 以子序列为单位的时序数据间的相似性可以通过快速傅里叶变换计算得到,大大减少了距离计算带来的时间消耗. 在获得 Shapelets 集合之后,利用这些 Shapelets 将原时序数据集转换成相应的特征向量表示,使用 SVM 分类器完成分类工作. 本文在多个领域的真实数据集上进行一系列实验,对比了目前主要 Shapelets 提取算法. 实验结果表明:本文提出的 Shapelets 提取算法具有很好的时间效率,同时,又能保证很高的分类准确率,与其他算法相比具有更好的实用性.

1 相关工作

目前的研究工作中对于 Shapelets 的提取思路主要有 2 种:基于搜索的策略[13-14,16-17,20-23] 和基于学习的策略[24-25].

1.1 基于搜索策略的 Shapelets 提取

在 Shapelets 提取最初的研究中,其基本思路是考虑训练数据中的所有子序列,然后使用信息增益(information gain, IG)来评估每一个子序列对于类别的区分能力,通过循环获取当前训练数据中具有最优信息增益度量的子序列来构建决策树分类器,实现时序数据的分类[13]. 类似于信息增益度量,一些新的度量方式,如 F-Statistic, Kruskall-Wallis 等,也被应用于衡量子序列的区分能力,但从效果上来看,信息增益要优于这些度量方式[17].

由于时序数据子序列数目众多,逐一进行判断的 brute-force 算法需要大量的运行时间,其时间复杂度为 $O(n^2m^4)$(n 为训练集中时序数据的条数),因此,相应的剪枝方法和加速技术是研究的重点. 基于最小距离的提前终止计算和基于信息增益的熵剪枝是最早提出的策略[13],这些策略虽然在一定程度上可以加速搜索过程,但仍然十分耗时. Mueen 等人[14] 提出以空间换时间的方法,使用 5 个充分统计量使得在距离计算过程中可以部分重用之前的结果. 另外,使用新的度量方式(信息增益上界)来减少耗时的熵的计算. 通过这些技术,算法的时间复杂度被减少至 $O(n^2m^3)$. 在文献[14]的基础上,原继东等人[20] 提出利用 Shapelets 之间的逻辑组合关系来提高分类准确率,但并没有提升 Shapelets 提取效率. 为进一步压缩搜索空间,Rakthanmanon 等人[16] 借助于时序数据符号化表示,将时序数据离散化,在低维空间中通过随机映射方法快速过滤掉类别区分能力低的子序列. 对于筛选出的有可能成为 Shapelets 的子序列,则继续使用信息增益加以判断. 该方法能够大大减小搜索空间,使得时间复杂度降低至 $O(nm^2)$.

除了使用加速技术改进算法外,其他方式的策略也被用于加快 Shapelets 提取. Chang 等人[21] 借助 GPU 运算对不同的子序列使用并行化方法来判断子序列分类能力. 然而,这一方面需要对距离计算方式做较大的修改,另一方面,其 Shapelets 的搜索时间非常依赖于硬件性能. Wistuba 等人[22] 基于有区分能力的 Shapelets 应当在时序数据中经常出现的假设,使用随机采样的方式构成 Shapelets. 这种策略确实可以减少运行时间,但分类准确率相对较低. Zhang 等人[23] 提出没有数值变化的子序列成为 Shapelets 的可能性很小,只利用有数值变化的子序列构成 Shapelets 候选空间. 这种方法大大减小搜索空间,有很好的加速效果,但本质上并没有改变算法时间复杂度.

1.2 基于学习策略的 Shapelets 提取

与搜索整个子序列空间不同,基于学习的策略试图寻找针对 Shapelets 提取的数学表示作为时序数据分类的目标函数,通过优化算法从时序数据集中学习到 Shapelets. 以这种方式得到的 Shapelets 有可能不是时序数据中的子序列,但确能将不同类别的时序更好地区分. Grabocka 等人[24]使用 logistic 回归分类模型构建 Shapelets 提取的目标函数,称为 LTS(learning time series),以随机梯度的方式不断地优化目标函数,使损失将至最低,进而构造出 Shapelets. 该方法得到的 Shapelets 往往不是某条时序数据中的子序列,而是能够对数据进行划分的特征. 目前,LTS 仍然是在不使用集成策略前提下的分类效果最好的算法[26]. 但是这种策略的最大问题是时间和空间消耗都很大,在硬件资源受限情况下无法得到分类结果. 另一种基于学习的方法是由 Hou 等人[25]提出的 FLAG(fused lasso generalized eigenvector method)算法,与 LTS 直接优化产生 Shapelets 不同,FLAG 通过构建目标函数来寻找具有好的分类能力的数据所在的位置信息,然后提取这些位置上的子序列形成 Shapelets. 这种方法的优点是速度快,但是分类准确率有一定的降低.

2 相关定义与符号表示

定义 1. 时序数据. 一条时序数据是一组实数值变量组成的向量,表示为 $\boldsymbol{T}=(t_1,t_2,\cdots,t_m)$,其中 m 为时序数据的长度.

定义 2. 子序列. 给定长度为 m 时序数据 $\boldsymbol{T}$,子序列 $\boldsymbol{S}$ 是 $\boldsymbol{T}$ 中的一个连续片段,记为 $\boldsymbol{S}=(t_p,t_{p+1},\cdots,t_{p+l-1})$,其中 p 为子序列在 $\boldsymbol{T}$ 中的起始位置,l 为子序列长度,$1\leqslant p\leqslant m-l+1$.

长度为 m 的时序数据 $\boldsymbol{T}$ 中包含 $m-l+1$ 条长度为 l 的子序列,可以使用滑动窗口逐一取出.

定义 3. 全子序列集合. 时序数据 $\boldsymbol{T}$ 的全子序列集合是指由长度为 l 的滑动窗口在 $\boldsymbol{T}$ 上生成的子序列集合,记为 $A=\{\boldsymbol{S}_{1,l},\boldsymbol{S}_{2,l},\cdots,\boldsymbol{S}_{m-l+1,l}\}$. 为了表示方便,本文使用 $\boldsymbol{A}[i]$ 表示 $\boldsymbol{S}_{i,l}$.

时序数据间的相似性通常是通过距离大小来度量的,长度相等的 2 条时序数据 $\boldsymbol{T}$ 与 $\boldsymbol{R}$ 之间的距离

$$d(\boldsymbol{T},\boldsymbol{R})=\sqrt{\frac{1}{m}\sum_{i=1}^{m}(t_i-r_i)^2}. \tag{1}$$

式(1)只能度量长度相等的时序数据,而本文主要的处理对象是子序列,也就是在不等长情况下度量序列间的相似性,为此,需要定义子序列与时序数据间的距离表示.

定义 4. 子序列与时序数据的距离. 给定时序数据 $\boldsymbol{T}$(全子序列集合为 A)及子序列 $\boldsymbol{S}$,两者之间的距离定义为子序列 $\boldsymbol{S}$ 在 $\boldsymbol{T}$ 中最优匹配时的欧氏距离:

$$sd(\boldsymbol{T},\boldsymbol{S})=\min(d(\boldsymbol{A}[1],\boldsymbol{S}),d(\boldsymbol{A}[2],\boldsymbol{S}),\cdots,d(\boldsymbol{A}[m-l+1],\boldsymbol{S})). \tag{2}$$

图 1 直观地描述了距离 $sd(\boldsymbol{T},\boldsymbol{S})$中最小值的含义.

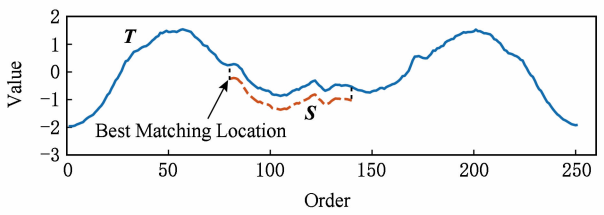


Fig. 1 Illustration of best matching in $\boldsymbol{T}$ for subsequence $\boldsymbol{S}$

图 1 子序列 $\boldsymbol{S}$ 在 $\boldsymbol{T}$ 中的最优匹配示例

尽管式(2)易于理解,但由于子序列数量庞大,其计算需要大量的运行时间,Yeh 等人[19]提出借助均值、方差等充分统计量和点积运算来降低计算的复杂度,即 $\boldsymbol{S}$ 与 $\boldsymbol{T}$ 中每一条子序列之间的计算为

$$d(\boldsymbol{A}[i],\boldsymbol{S})=\sqrt{2l\left(1-\frac{\boldsymbol{A}[i]\cdot\boldsymbol{S}-l\mu_{\boldsymbol{A}[i]}\mu_{\boldsymbol{S}}}{l\sigma_{\boldsymbol{A}[i]}\sigma_{\boldsymbol{S}}}\right)}, \tag{3}$$

其中,$\mu_{\boldsymbol{A}[i]},\mu_{\boldsymbol{S}},\sigma_{\boldsymbol{A}[i]},\sigma_{\boldsymbol{S}}$ 分别为子序列 $\boldsymbol{A}[i],\boldsymbol{S}$ 的均值和方差. 通常情况下,对于每一条子序列,均值和方差的计算时间复杂度为 $O(l)$,但可以通过存储 $\boldsymbol{T}$ 中数据值和数据值平方的累积和向量,使得在每一次运算中都可以直接用来计算任意长度的子序列的均值和方差[27]. 这样,在式(3)中,$\boldsymbol{A}[i]$ 与 $\boldsymbol{S}$ 的点积运算就成为制约计算效率的关键因素,3.2.1 节将给出快速计算该点积操作的算法.

定义 5. 最近邻函数. 给定 2 条时序数据的全子序列集合 A 和 B,对于任意一子序列对$\langle\boldsymbol{A}[i],\boldsymbol{B}[j]\rangle$,如果 $\boldsymbol{A}[i]$在 B 中的最优匹配是 $\boldsymbol{B}[j]$,则最近邻函数 θ 值为 1,否则为 0,即:

$$\theta(\boldsymbol{A}[i],\boldsymbol{B}[j])=\begin{cases}1, & d(\boldsymbol{A}[i],\boldsymbol{B}[j])\leqslant d(\boldsymbol{A}[i],\boldsymbol{B}[k]),\\ 0, & \text{otherwise},\end{cases} \tag{4}$$

其中,$k\in[1,m-l+1]$且 $k\neq j$. 使用最近邻函数,可以得到 2 条时序数据的相似性连接向量.

定义 6. 相似性连接向量. 给定 2 条时序数据的全子序列集合 A 和 B,对于 A 中的每一条子序列 $\boldsymbol{A}[i]$,使用最近邻函数在 B 中寻找其最优匹配 $\boldsymbol{B}[j]$,每一个匹配对之间的距离所构成的向量称为 A 与 B 之间的相似性连接向量,记为 $\boldsymbol{P}_{AB}$.

需要注意的是,$\boldsymbol{P}_{AB}$是有序的,在计算过程中严格按照第 1 条时序的全子序列集合 A 中子序列的先后顺序寻找在 B 中的最近邻,也就是说 A 中的子序列一定会在$\theta=1$的最优匹配对中出现,而 B 中的子序列不一定全部在 $\theta=1$ 的匹配对中,因此,$\boldsymbol{P}_{AB}$并不是对称的,即 $\boldsymbol{P}_{AB}\neq\boldsymbol{P}_{BA}$.

定义 7. 自相似性连接向量. 在计算 $\boldsymbol{P}_{AB}$时,若 $B=A$,这样构成的相似性连接向量称为自相似性连接向量,记为 $\boldsymbol{P}_{AA}$.

$\boldsymbol{P}_{AA}$可看作表示一条时序数据的元数据. 由于每一条子序列都来自该时序数据,所以总能找到距离为 0 的最优匹配以及在最优匹配附近近似等于 0 的匹配子序列,这样的匹配对于描述数据特性是没有意义的,通常称之为平凡匹配(trivial matching)[28]. 因此,在计算过程中,需要避免这样的匹配. 2 条时序数据之间的相似性可以通过 $\boldsymbol{P}_{AA}$和 $\boldsymbol{P}_{AB}$表现出来,为此,我们定义差异向量来描述时序数据之间的差异.

定义 8. 差异向量. 对于 2 条不同时序数据的全子序 A 和 B,则两者的差异向量 $\boldsymbol{Diff}_{AB}$定义为以 A 为基准的相似性连接向量 $\boldsymbol{P}_{AB}$和 A 的自相似性连接向量 $\boldsymbol{P}_{AA}$之间差的绝对值,形式化表示为:若

$$\begin{aligned}\boldsymbol{P}_{AB}&=(d_1^{AB},d_2^{AB},\cdots,d_{m-l+1}^{AB}),\\ \boldsymbol{P}_{AA}&=(d_1^{AA},d_2^{AA},\cdots,d_{m-l+1}^{AA}),\end{aligned}\tag{5}$$

则 $\boldsymbol{Diff}_{AB}=(|d_1^{AB}-d_1^{AA}|,\cdots,|d_{m-l+1}^{AB}-d_{m-l+1}^{AA}|)$.

在时序数据集中,如果同一类的时序数据之间在 $\boldsymbol{Diff}_{AB}$的某些分量上值比较小,而在不同类的数据上值比较大,则可认为相应的子序列是具有较高的类别可分性,是比较好的 Shapelets. 在第 3 节中,将详细描述如何使用这些定义提取高质量的 Shapelets.

3 Shapelets 提取

本文所提出的基于相似性连接的时序数据 Shapelets 提取主要由 3 部分构成:1)数据的预处理过程,主要完成对时序数据集的精简工作,只保留对 Shapelets 提取起关键作用的训练数据以加速后续处理;2)通过计算相似性连接向量、自相似性连接向量及差异向量来生成 Shapelets 候选矩阵,然后从候选矩阵中提取相对应的子序列,并对这些子序列进行合并操作实现 Shapelets 提取,这是本文的核心;3)完成时序数据的转换工作,借助已提取的 Shapelets 将时序数据转换为由距离值构成的特征向量表示. 对于时序数据的特征向量表示,使用现有成熟的分类器,如 SVM,完成分类工作. 本文方法的整体框架及流程如图 2 所示:

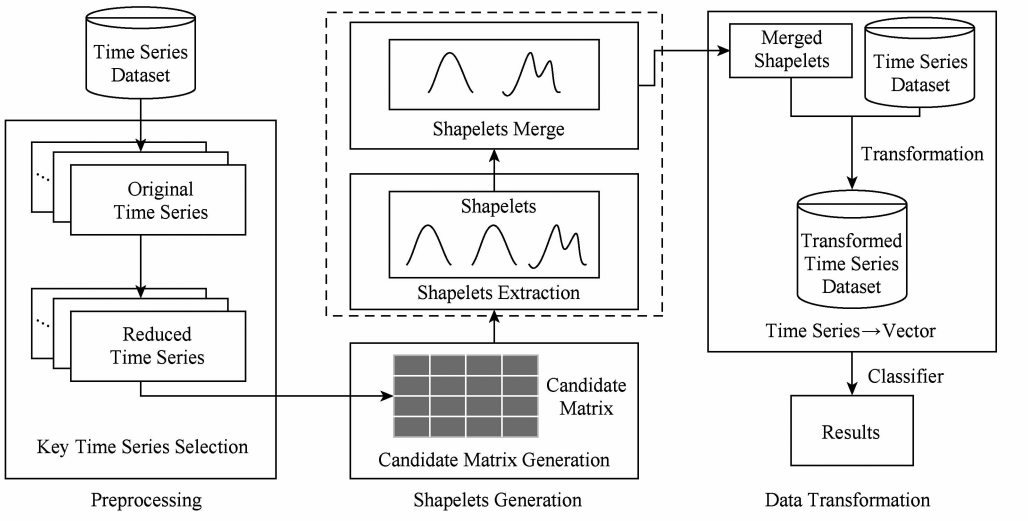


Fig. 2 The overall framework and whole process of the proposed method

图 2 本文方法的整体框架及流程

3.1 关键时序数据的选择

在时序数据集中,往往存在着一些高度相似的数据,其全子序列集合也非常相近. 由于使用相似性连接向量计算差异向量的过程相对复杂,因此,对原始时序数据集进行精简,只选择相似度比较小的关键时序数据进行计算是十分必要的. Wistuba 等人[22]基于 Shapelets 在时序数据集中出现的频率高的假设所做的工作也验证了只选用关键时序数据提取高质量的 Shapelets 是可行的.

由于在选择关键时序数据时不仅要求整体上有

极大的相似度,还需要子序列之间也要高度相似,因此本文使用不具有任何偏移处理的欧氏距离(式(1))计算时序数据间的相似程度.通常情况下,同类别中存在高相似性的数据的可能性大,而不同类之间,时序数据的差异较大.基于此,本文按类别分别进行关键时序数据的选择.对于每一类中的时序数据,使用队列作为核心数据,计算队首时序与其他时序的距离,若距离小于预先设定的阈值,则认为两者有高度相似性,然后提取队首时序作为关键时序数据,而与队首时序相似的时序就可以舍弃.具体算法如算法 1 所示:

算法 1. 关键时序选取算法.

输入:时序数据集 *dataSet*、相似度阈值 *mthres*;

输出:关键时序数据集 *nDataSet*.

① *tempDataSet*=*partitionByClasses*(*dataSet*);
② *nDataSet*=[];
③ for each *classData* in *tempDataSet*
  /* 每一类 */
④ *queue*=*index*(*classData*); /* 使用队列存储当前类时序的索引 */
⑤ while *queue* is not empty
⑥ *pos*=*distance*(*classData*,*queue*,*mthres*);
  /* 查找当前类别中与队首数据距离小于阈值的时序的位置 */
⑦ *queue*=*remove*(*queue*,*pos*);
  /* 删除队列中 *pos* 位置的索引 */
⑧ *nDataSet*=[*nDataSet*;*classData*(*queue*[1])];
⑨ end while
⑩ end for

算法 1 所消耗的运行时间是很少的.首先,算法在选取关键时序数据时是以类别为单位的,不同类的时序数据间不进行距离计算,这使得参与相似度计算的时序数据条数大大减少.其次,在每次迭代过程中,算法在只保留 1 条关键时序数据的同时,从队列中删除与之相似的数据,减少了下次迭代过程中参与计算的时序数量.具体地说,若时序数据集(大小为 n)的第 i 类中含有 n_i 条时序,则在最好情况下,所有时序均相似,每一条时序只参与一次距离计算(n_i-1 次);在最坏情况下,所有时序均不相似,需进行 $n_i(n_i-1)/2$ 次距离计算,故算法 1 在处理第 i 类时序数据时距离计算的次数介于 n_i-1 至 $n_i(n_i-1)/2$ 之间.因此,算法 1 的距离计算次数约为 $n\sim n(n_i-1)/2$.通常情况下,n_i 要比 n 小得多且数据集中存在大量的相似时序数据,所以算法 1 具有很高的时间效率.

3.2 候选矩阵

候选矩阵(candidate matrix)是本文基于相似性连接策略提取时序数据 Shapelets 的核心,候选矩阵中隐含着可用于表示类别差异的 Shapelets,其值表示相连接的时序在相应子序列上的差异,值越大,对应子序列区分能力越强,因此,可从候选矩阵中快速定位并提取出高类别可分性的子序列.候选矩阵生成的时间消耗是影响算法整体效率的关键因素,如何快速的生成候选矩阵是本文的重点研究内容.

计算候选矩阵的关键是相似性连接,而相似性连接属于全匹配问题,即需要计算出时序数据全子序列集合中每一个子序列在其他集合中的最近邻子序列.候选矩阵的计算分为 3 个部分:1)计算距离向量,即子序列与时序数据全部子序列间的欧氏距离;2)根据相似性连接策略计算两条时序数据全子序列集合间的最优匹配,得到两者之间的相似性连接向量;3)利用 2 条时序数据的相似性连接向量和其中 1 条时序数据的自相似性连接向量构造差异向量,将由同一条时序数据的自相似性连接向量得到的差异向量合并,即可组成候选矩阵.

3.2.1 距离向量的快速计算

距离向量是某一子序列 S 与一条时序数据 T 全部子序列之间的欧氏距离构成的向量,其核心是 2 个子序列距离的计算问题.然而,由于子序列数量庞大,逐个计算子序列间的距离会消耗大量的运行时间,降低算法效率.为此,本文根据 Rakthanmanon 等人[27]提出的方法,使用式(3)的计算方式,实现子序列与时序数据全子序列集合之间距离的快速计算.式(3)中最关键部分是子序列之间内积的运算,Mueen 等人[29]已经证明,借助快速傅里叶变换,可以使得子序列与时序数据全部子序列之间的内积在 1 次计算中完成.距离向量的具体实现如下:

算法 2. 距离向量计算算法 *distanceVector*.

输入:时序数据 $\boldsymbol{T}$ 及长度 m、子序列 $\boldsymbol{S}$ 及长度 l;

输出:$\boldsymbol{S}$ 与 $\boldsymbol{T}$ 的距离向量 $\boldsymbol{D}$.

① $[\mu_T,\mu_S,\sigma_T,\sigma_S]=meanStd(\boldsymbol{S},\boldsymbol{T})$; /* 根据文献 0 计算均值方差 */
② $\boldsymbol{S}'=reverse(\boldsymbol{S})$; /* 将 $\boldsymbol{S}$ 翻转 */
③ $\boldsymbol{T}_a=append(\boldsymbol{T},m)$; /* 对 $\boldsymbol{T}$ 扩充,填 m 个 0 */
④ $\boldsymbol{S}'_a=append(\boldsymbol{S}',2m-l)$; /* 与 $\boldsymbol{T}_a$ 等长 */
⑤ $\boldsymbol{S}'_{af}=\mathrm{FFT}(\boldsymbol{S}'_a)$; /* 快速傅里叶变换 */
⑥ $\boldsymbol{T}_{af}=\mathrm{FFT}(\boldsymbol{T}_a)$;

⑦ $\boldsymbol{ST}_t$=MUL($\boldsymbol{S}'_{af}$,$\boldsymbol{T}_{af}$); /* 逐元素相乘 */

⑧ $\boldsymbol{ST}$=$inverseFFT(\boldsymbol{ST}_t)$;

⑨ $\boldsymbol{D}$ =$distance(l,\boldsymbol{ST},\mu_T,\mu_S,\sigma_T,\sigma_S)$. /* 根据式(3)计算距离并组成距离向量 */

算法行⑧得到的 $\boldsymbol{ST}$ 是子序列 $\boldsymbol{S}$ 与 $\boldsymbol{T}$ 中每一个子序列点积所构成的有序向量,即 $ST[i]$表示 $\boldsymbol{S}$ 与 $\boldsymbol{T}$ 中第 i 条子序列 $\boldsymbol{A}[i]$的点积. 因此,在使用式(3)计算距离向量(行⑨)时,只需要从中取出相应的值,而不用每次都计算内积. 该算法的一个优势在于算法的时间复杂度与子序列长度无关,这也就不存在最好情况与最坏情况的差异,其主要时间消耗在于傅里叶变换($O(m\,\mathrm{lb}\,m)$),故算法的运行效率高.

3.2.2 提取相似性连接向量

对于 2 条时序数据 $\boldsymbol{T}$,$\boldsymbol{R}$(其全子序列集合分别为 A,B),可以迭代使用算法 2 计算 A 中每一条子序列与 $\boldsymbol{R}$ 的距离向量. $\boldsymbol{T}$ 和 $\boldsymbol{R}$ 的相似性连接向量 $\boldsymbol{P}_{AB}$ 就可以通过查找这些距离向量的最小值得到. 在计算过程中,本文采用在每次计算距离向量之后更新 $\boldsymbol{P}_{AB}$ 的方式,逐步得到与 $\boldsymbol{T}$ 中每一条子序列最优匹配的 $\boldsymbol{R}$ 中的子序列,记录其距离和位置,具体算法如下:

算法 3. $\boldsymbol{P}_{AB}$ 提取算法 *similarityJoinVector*.

输入:时序数据 $\boldsymbol{T}$,$\boldsymbol{R}$ 及长度 m、子序列长度 l;

输出:$\boldsymbol{P}_{AB}$.

① B=$allSubseqSet(\boldsymbol{R})$;
/* $\boldsymbol{R}$ 的全子序列集合 */

② initialize $\boldsymbol{P}_{AB}$;

③ $index$=1; /* B 中子序列索引变量 */

④ for each $\boldsymbol{S}\in B$

⑤ $\boldsymbol{D}$=$distanceVector(\boldsymbol{T},m,\boldsymbol{S},l)$;
/* 算法 2 */

⑥ $\boldsymbol{P}_{AB}$=$updating(\boldsymbol{P}_{AB},\boldsymbol{D},index)$;
/* 逐元素更新距离值 */

⑦ end for

在算法 3 中,$\boldsymbol{P}_{AB}$ 是以 $\boldsymbol{T}$ 为基准计算的,每一次的迭代中,算法都更新 $\boldsymbol{T}$ 相应位置上的距离值,直至 B 中所有子序列计算完成. 这种更新方式的好处是不必开辟内存空间去存储所有子序列对之间的距离,减少了空间消耗. 如果算法 3 的输入中 $\boldsymbol{R}$ 替换为 $\boldsymbol{T}$,则算法得到的是 $\boldsymbol{T}$ 的自相似性连接向量 $\boldsymbol{P}_{AA}$,但必须要处理平凡匹配导致的距离为 0 的问题. 本文采用 Yeh 等人[19]的处理方式,直接跳过相关区域的计算,且不更新 $\boldsymbol{P}_{AA}$ 中对应位置上的值. 图 3 直观描述了算法 3 生成 $\boldsymbol{P}_{AA}$ 的过程. 当 $\boldsymbol{P}_{AA}$ 初始化为 Inf 后,按顺序计算每一条子序列与 $\boldsymbol{T}$ 的距离向量并更新 $\boldsymbol{P}_{AA}$,为了方便展示,本文只选取第 i 条和第 j 条子序列作为示例来更新 $\boldsymbol{P}_{AA}$. 在得到 $\boldsymbol{A}[i]$与 $\boldsymbol{T}$ 的距离向量(以 $\boldsymbol{D}_i$ 表示)之后,按照逐元素取最小值的策略更新 $\boldsymbol{P}_{AA}$ 对应位置上的值(图 3 中第 1 行和第 2 行),得到更新后的 $\boldsymbol{P}_{AA}$. 当遇到平凡匹配的情况时,即距离向量中值为 0 的情况,则保留 $\boldsymbol{P}_{AA}$ 原来的值. 对于 $\boldsymbol{A}[j]$,思路相同,所不同的是生成的距离向量(以 $\boldsymbol{D}_j$ 表示)比较的对象是上次更新后的 $\boldsymbol{P}_{AA}$(图 3 中第 3 行和第 4 行).

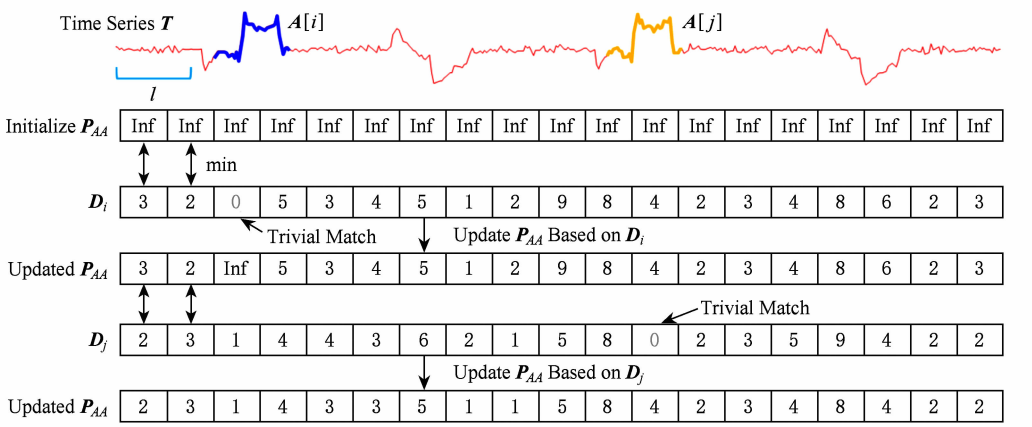


Fig. 3 The process of $\boldsymbol{P}_{AA}$ updating

图 3 $\boldsymbol{P}_{AA}$ 更新过程

3.2.3 候选矩阵生成

由 3.2.2 节可知,$\boldsymbol{P}_{AB}$ 是以子序列为单位描述两时序数据的,其相似性可以通过 $\boldsymbol{P}_{AB}$ 与 $\boldsymbol{P}_{AA}$ 的差值来体现,即差异向量 $\boldsymbol{Diff}_{AB}$. 然而由于 Shapelets 是具有类别可分性的子序列,体现的是类别差异,因此,需要考虑数据集中的全部时序数据,将得到的差异向量进行综合分析,为此,本文将所得到的差异向量进行组合,形成候选矩阵,用于后续处理.

一个关键的问题是并不是所有时序数据之间的差异向量对于提取 Shapelets 都是有用的. 如果 2 条时序数据具有相同的类别标识,其差异向量描述的是两者在数据生成过程中的差异,并不是体现不同类别时序数据的本质区别. 因此本文以类别为依据,在计算差异向量时,从不同类别中选取时序数据,而同类别时序数据间不进行计算. 若某一时序数据集中包含 3 个类别,分别为Ⅰ,Ⅱ,Ⅲ类,差异向量的计算过程为:从Ⅰ类中选出 1 条时序数据 $\boldsymbol{T}$,以 $\boldsymbol{T}$ 为基准依次计算其与Ⅱ类和Ⅲ类中所有时序数据的差异向量. 对 $\boldsymbol{T}$ 与每一类时序数据得到差异向量进行逐点求平均值,得到平均差异向量,将其合并到候选矩阵中,之后对Ⅰ类中的其他数据进行相同的操作. 然后再以Ⅱ类中时序数据为基准计算其与Ⅲ类中时序数据的平均差异向量. 本质上,平均差异向量从统计意义上表示了 $\boldsymbol{T}$ 与某一类别时序数据间的不同. 候选矩阵生成的具体算法过程如下:

算法 4. 候选矩阵生成算法.

输入:关键时序数据集 *nDataSet*、子序列长度 l;

输出:候选矩阵 $\boldsymbol{M}$.

① *tempDataSet* = *partitionByClasses*(*nDataSet*);
/ * 按类别划分关键时序数据集 * /
② initialize $\boldsymbol{M}$;/ * 初始化候选矩阵 * /
③ for each *currClass* in *tempDataSet*
④ for each $\boldsymbol{T}\in$ *currClass*
⑤ $\boldsymbol{P}_{AA}$ = *similarityJoinVector*($\boldsymbol{T},\boldsymbol{T},l$);
⑥ initialize $\boldsymbol{Z}$; / * 初始化差异矩阵 * /
⑦ for each *otherclass* after *currClass*
⑧ for each $\boldsymbol{R}\in$ *otherclass*
⑨ $\boldsymbol{P}_{AB}$ = *similarityJoinVector*($\boldsymbol{T},\boldsymbol{R},l$);
⑩ $\boldsymbol{Z}=[\boldsymbol{Z};\mathrm{abs}(\boldsymbol{P}_{AB}-\boldsymbol{P}_{AA})]$;
⑪ end for
⑫ end for
⑬ $\boldsymbol{M}=[\boldsymbol{M};sum(\boldsymbol{Z})/\boldsymbol{Z}.rows]$;
⑭ end for
⑮ end for

在候选矩阵生成过程中,需要对数据集中所有的时序对计算 $\boldsymbol{P}_{AB}$ 和 $\boldsymbol{P}_{AA}$,算法 4 本身的复杂度较高,但由于本文在预处理阶段借助算法 1 大大减小了数据集规模,仅使用关键时序数据用于计算,因此,在实验中不会消耗很多的运行时间. 候选矩阵中的每一行代表着 1 条时序数据与另一类时序数据的平均差异,使用平均值的目的在于剔除这一类时序数据之间的差异,只使用共性的特征来提取 Shapelets.

3.3 基于候选矩阵的 Shapelets 提取

从 3.2 节可知,候选矩阵中的每一个值表示 1 条子序列与 1 类时序数据中最优匹配的差异,其值的大小代表该子序列区分能力,因此,Shapelets 可以通过提取候选矩阵中较大的值对应的子序列得到. 为了更形象地描述 Shapelets 提取过程,本文首先以 2 条时序数据为例,展示其相似性连接向量、自相似性连接向量及差异向量的计算过程,提取出能够表示两者之间差异的子序列,然后将其扩展到整个时序数据集上. 在差异向量上选取可能成为 Shapelets 的子序列时,阈值的选择对于 Shapelets 的数量有很大影响. 阈值设置过大,会导致仅有少数子序列构成 Shapelets,不能够全面体现时序间的差异;阈值过小,则会使得大量可区分性不明显的子序列也被加入到 Shapelets 中,这一方面会导致后续处理计算量的增加,另一方面会对分类过程形成干扰. 在实验中,本文通过交叉验证的方式来确定合适的阈值. 时序数据的 Shapelets 提取地过程如图 4 所示.

2 条示例时序数据 $\boldsymbol{T}$ 和 $\boldsymbol{R}$ 选自于 UCR archives[①] 的 UWaveGestureLibraryAll 时序数据集,且属于不同的类别,如图 4(a)所示. 假设 $\boldsymbol{T}$ 和 $\boldsymbol{R}$ 的全子序列集合分别为 A 和 B,则由算法 2 得到的 $\boldsymbol{P}_{AB}$ 与 $\boldsymbol{P}_{AA}$ 如图 4(b)所示. 从图 4(b)中可以看出,$\boldsymbol{P}_{AB}$ 与 $\boldsymbol{P}_{AA}$ 之间在某些位置上存在着明显的差异,这些差异所对应的子序列就可以将 $\boldsymbol{T}$ 和 $\boldsymbol{R}$ 进行区分. 图 4(c)给出了两者的差异向量,根据其值的大小可以很容易得到有区分能力的子序列所在的位置,如图 4(c)中的点 P,然后在原始时序数据 $\boldsymbol{T}$ 中即可将相应的子序列 $\boldsymbol{A}[P]$ 提取出来. 在构成 Shapelets 时,若子序列位置之间的距离小于子序列长度(如图 4(c)中以 P 和 Q 开始的子序列),本文将其合并形成 Shapelets,如图 4(d)所示.

如果由算法 1 所提取的关键时序数据集中只包含 2 类数据且每类中仅有 1 条时序数据,按照图 4 的流程就是 Shapelets 提取的全过程,差异向量 $\boldsymbol{Diff}_{AB}$ 即为候选矩阵. 但在一般情况下,数据集中会包含多类,而每类中含有若干时序数据,因此在提取 Shapelets 时,需要逐行扫描候选矩阵,在每一条时序上均提取子序列,实现算法如下:

① http://www.cs.ucr.edu/~eamonn/time_series_data/

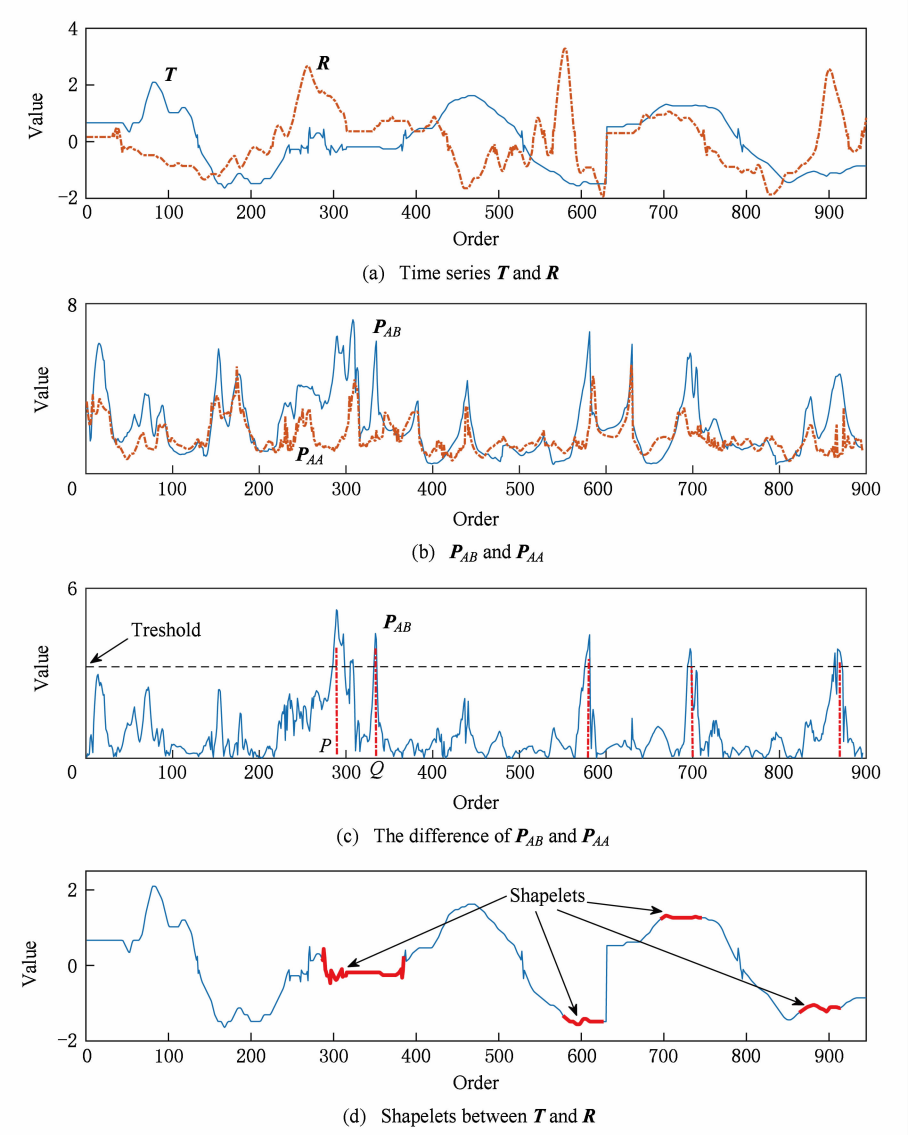


Fig. 4 The process of Shapelets extraction between $\boldsymbol{T}$ and $\boldsymbol{R}$

图 4 时序数据 $\boldsymbol{T}$ 和 $\boldsymbol{R}$ 的 Shapelets 提取过程

算法 5. Shapelets 提取算法.

输入:候选矩阵 $\boldsymbol{M}$、子序列长度 l、阈值 th、时序数据集 $DataSet$;

输出:Shapelets 集合 SL.

① initialize SL; /* 初始化 Shapelets 集合 */

② for each row in $\boldsymbol{M}$

③ initialize $tempSL$;
/* 二元组(位置,子序列) */

④ for $i=1:length(row)$

⑤ if $row[i]>th$

⑥ $subSequence=DataSet(row,i:i+l-1)$; /* 提取子序列 */

⑦ $tempSL=[tempSL;[i,subSequence]]$;

⑧ end if

⑨ end for

⑩ $tSL=mergeSubsequence\ (tempSL)$;
/* 将有重合部分的子序列合并 */

⑪ $SL=[SL;tSL]$;

⑫ end for

算法 5 与图 4 所描述过程本质上是一样的,唯一的区别在于候选矩阵值的计算方式有所不同. 假设时序数据集中包含 3 类:Ⅰ,Ⅱ,Ⅲ类,则多类情况下的 Shapelets 提取基本过程如图 5 所示. 图 5 中第Ⅰ类所提取的 Shapelets,可以将Ⅰ类时序数据从数据集中区分开来,但并不能将Ⅱ类和Ⅲ类的时序数据进行区分,因此还需要以Ⅱ类时序数据为基准从Ⅱ类中提取 Shapelets,此时,Ⅰ类中的时序数据不再参与计算以减少时间消耗.

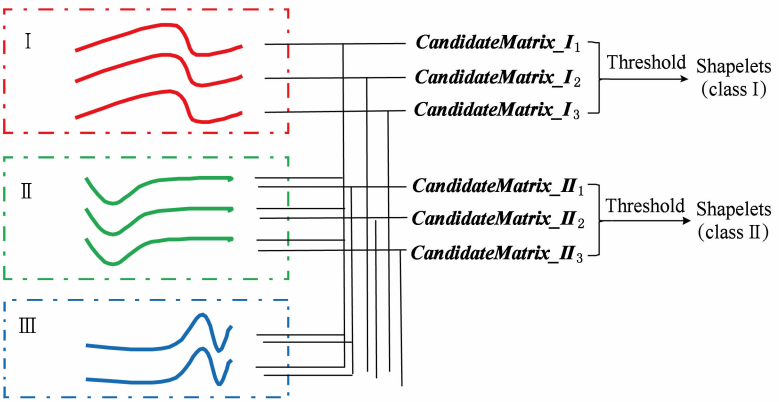


Fig. 5 The process of shapelets extraction

图 5 Shapelets 提取过程

3.4 算法分析

从算法 2～5 描述过程可以看出,本文所提出的 Shapelets 提取算法以不同类别时序的差异为基本出发点,从时序数据中提取能够区分不同类别的子序列是一种最直观的方式. 与现有 Shapelets 提取算法不同的是本文直接从子序列的角度描述时序数据间的相似性,其优势在于易于发现时序间差异所对应的子序列. 研究的核心在于如何以子序列为基准快速地计算不同类别时序间的差异. 本文通过 2 个方面来加速计算:1)通过选取关键时序数据来减少后续参与计算的时序数量,通过实验验证(5.2 节)该步骤是合理和有效的;2)借助傅里叶变换快速完成时序间的相似性连接向量和自相似性连接向量. 本质上,本文是通过两两比较来构成 Shapelets 集合的,这样得到的每一条子序列至少能区分来自不同类别的 2 条时序,由其构成的集合也能够很好的将不同类别的时序区分开来,因此,算法是合理可行的.

4 时间序列空间变换

在以 Shapelets 为特征的时序数据分类算法中,决策树是使用最为广泛的分类器,其优点是容易构造,对分类结果有直观的解释,但缺点也很明显. 决策树以 Shapelets 为节点,将数据集不断地进行划分,直至每个子数据集不能再分或者达到某些终止条件. 这会导致 2 方面的问题:如果只使用少数 Shapelets 构建浅层二叉树,其分类准确率受到限制,并且很多 Shapelets 具有非常相近的分类能力,不恰当的选择也会导致分类准确率降低;而如果使用的 Shapelets 太多,树的结构复杂,又会导致运行时间加大. Bagnall 等人[30]指出将时序数据映射到特征易于被检测的空间是解决时序数据分类问题的一个好方法,本文借鉴 Lines 等人[18]的策略,将时序数据映射到 Shapelets 所构成的空间,形成原始时序数据的特征表示.

图 6 描述了基于距离运算的空间转换的方法:

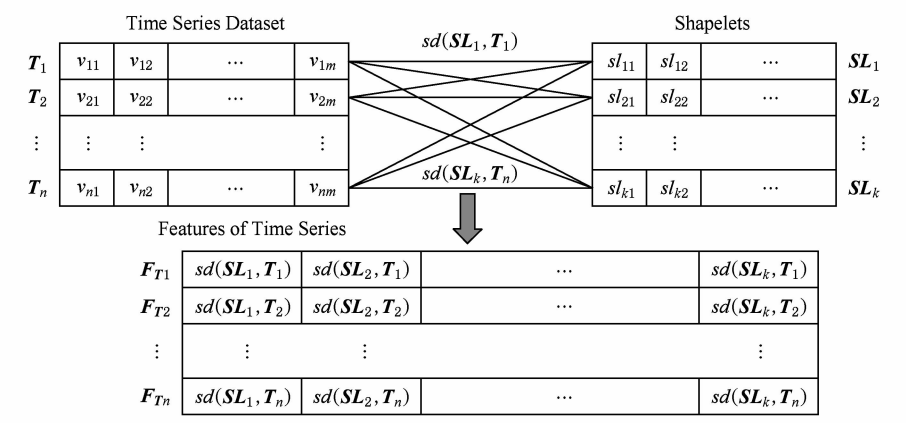


Fig. 6 The space transformation of time series

图 6 时序数据的空间变换示意图

假设提取的 Shapelets 数目为 k,对于每一条时序数据 $\boldsymbol{T}_i$,依次计算其与 k 条 Shapelets($\boldsymbol{SL}_1$,$\boldsymbol{SL}_2$,…,$\boldsymbol{SL}_k$)之间的距离(式(2)),将这些距离组合起来形成 1 个距离向量,该距离向量即为原时序数据在 Shapelets 空间中的表示. 这种表示一方面保持了 Shapelets 在时序数据分类问题中的优势,因为向量中每一个值都是以 Shapelets 为单位得到的;另一方面,向量中的每个值都蕴含着时序数据中数据的先后次序关系,因此,向量中各元素之间具有相互独立性,可以使用现有成熟的分类器(本文使用 SVM)进行分类.

5 实验与分析

本文通过对时序数据进行分类来验证所提出的 Shapelets 提取方法的有效性. 本节将首先介绍相关的实验设置,包括实验所使用的时序数据集、评价指标以及用于比较的 Shapelets 提取算法,之后给出实验结果并进行分析和讨论.

5.1 实验设置

5.1.1 数据集

本文使用 UCR archives 和 UEA 时序数据集① 中的 26 个数据集来评估方法的性能,这些数据集是目前 Shapelets 相关研究中普遍使用的数据集[16,25]. 数据来自于多个领域,包括传感器数据、图像轮廓信息、人体心电图以及动作数据等,数据集的基本信息如表 1 所示:

Table 1 Datasets Used in the Experiments

表 1 实验中所使用的数据集

Datasets	# Train	# Test	Length	# Classes
Adiac	390	391	176	37
Beef	30	30	470	5
Chlorine.	467	3 840	166	3
Coffee	28	28	286	2
Diatom.	16	306	345	4
DP_Little	400	645	250	3
DP_Middle	400	645	250	3
DP_Thumb	400	645	250	3
ECGFiveDays	23	861	136	2
FaceFour	24	88	350	4
Gun_Point	50	150	150	2
ItalyPower.	67	1 029	24	2
Lighting7	70	73	319	7
MedicalImages	381	760	99	10
MoteStrain	20	1 252	84	2
MP_little	400	645	250	3
MP_Middle	400	645	250	3
Otoliths	64	64	512	2
PP_little	400	645	250	3
PP_Middle	400	645	250	3
PP_Thumb	400	645	250	3
Sony	20	601	70	2
Symbols	25	995	398	6
SyntheticC	300	300	60	6
Trace	100	100	275	4
TwoLeadECG	23	1 139	82	2

从表 1 可以看出,所使用的数据集类型多样:从二分类到多分类,且多分类问题较多,最多可达 37 类(Adiac);长度不一,最短为 24(ItalyPower.),最长为 512(Otoliths);训练集和测试集的数据差异也比较大,因此,可以全面的度量算法的性能. 为了便于性能比较,本文采用默认的训练集和测试集划分,实验中相关的参数均通过交叉验证的方式获得.

5.1.2 方法对比

本文算法(similarity join for Shapelets extraction, SJS)与目前主要的 7 个 Shapelets 提取算法进行性能比较:

1) 标准基于 Shapelets 时序数据分类算法使用不同的度量指标:信息增益(IG)[13]、Kruskal-Wallis 检验(KW)[17]、F-统计量(FS)[17].

2) FSH(fast-Shapelets)[16] 算法. 将时序数据转换为离散的低维表示,在低维空间中过滤掉大部分子序列以加速搜索的算法.

3) LTS[24] 算法. 使用梯度下降法从数据中学习出最优 Shapelets 和分类决策函数的算法.

4) IGSVM[18] 算法. 首次对时序数据进行空间变换的算法,以信息增益作为 Shapelets 区分能力度量,使用线性 SVM 作为分类器完成分类工作.

5) FLAG[25] 算法. 采用学习优化的策略,使用广义特征向量的方法对时序数据进行降维,进而提取相应子序列作为 Shapelets 的算法.

上述对比算法的实现均由提出该方法的原作者提供,其参数为作者推荐的设置.

① http://www.uea.ac.uk/computing/machine-learning/shapelets/shapelet-data

5.1.3 评价指标

对于分类问题,分类准确率是评价算法性能最重要的标准,但对于多种算法在多个数据集上的结果比较,分类准确率难以直观地度量各个算法的性能,因为在很多情况下,某一算法在一数据集上效果很好,但在另一数据集上,分类结果可能较差.因此,本文采用在机器学习领域广泛使用的无参数 Friedman 测试作为评价算法性能的标准.本质上,Friedman 测试是基于分类准确率计算的.在获得 K 个算法在 N 个数据集上的分类结果后,对每一个数据集上的 K 个算法进行排序,分类准确率最高的算法标记为"1",次高标记为"2",以此类推,分类效果最差的算法标记为"K",这样就可以得到 $N\times K$ 的排序矩阵 $\boldsymbol{H}$,其中,h_{ij} 表示第 i 个数据集上第 j 个算法的标记值.然后,计算每个算法的平均标记值

$$\bar{h}_j = \sum_{i=1}^{N} h_{ij}/N. \tag{6}$$

在零假设(null hypothesis)的情况下,所有算法是等价的,因而 $\bar{h}_j$ 应当相等,所以当 N 和 K 足够大时(一般 $N>10$,$K>5$),Friedman 统计量:

$$\chi^2_{\text{Friedman}} = \frac{12N}{K(K+1)}\left[\sum_{j=1}^{K}\bar{h}_j - \frac{K(K+1)^2}{4}\right], \tag{7}$$

可以用具有 $K-1$ 个自由度的卡方分布近似.Demšar[31]通过分析前人的工作得出卡方分布是相当保守的近似,提出使用

$$F = \frac{(N-1)\chi^2_{\text{Friedman}}}{N(K-1)-\chi^2_{\text{Friedman}}}, \tag{8}$$

作为度量.当实验结果拒绝零假设时,Demša 提出将所有算法按照 F 值分到不同的组,使得同一组内的算法在分类准确率上没有显著差异.这样,不同算法的性能差异可以通过包含平均次序和无明显差异算法组的关键差异图(critical difference diagram)表示.

5.2 提取关键时序数据对实验结果的影响

本节首先验证仅仅使用关键时序数据所提取的 Shapelets 对于分类结果的影响.图 7 显示了在表 1 数据集上使用算法 1 和不使用算法 1 的情况下,所提取的 Shapelets 用于分类的结果对比,其中,N-SJS 表示使用全部时序数据时的算法,SJS 算法表示仅使用关键时序数据的算法.横、纵轴均以分类准确率为单位.

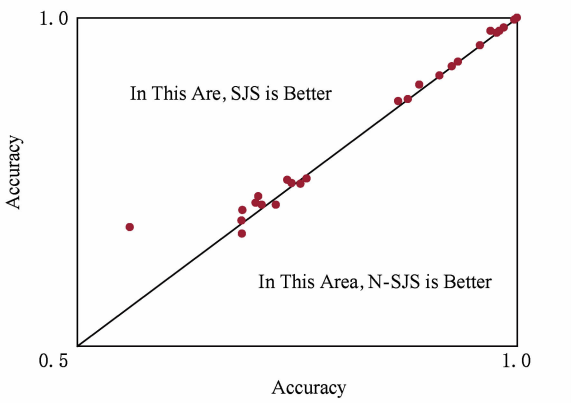


Fig. 7 Classification accuracy comparison of SJS and N-SJS algorithm for datasets in Table 1

图 1 SJS 算法和 N-SJS 算法在表 1 数据集上的分类准确率比较

从表 1 数据集上的分类结果(图 7)可以看出,在绝大多数数据集上,SJS 和 N-SJS 几乎没有区别,即分类结果位于中间对角线附近,说明通过算法 1 精简数据集后,并没有减弱本文方法提取高质量 Shapelets 的能力.同时,在部分数据集上,准确率略有提升,意味着冗余的 Shapelets 也可能导致分类器性能的下降.

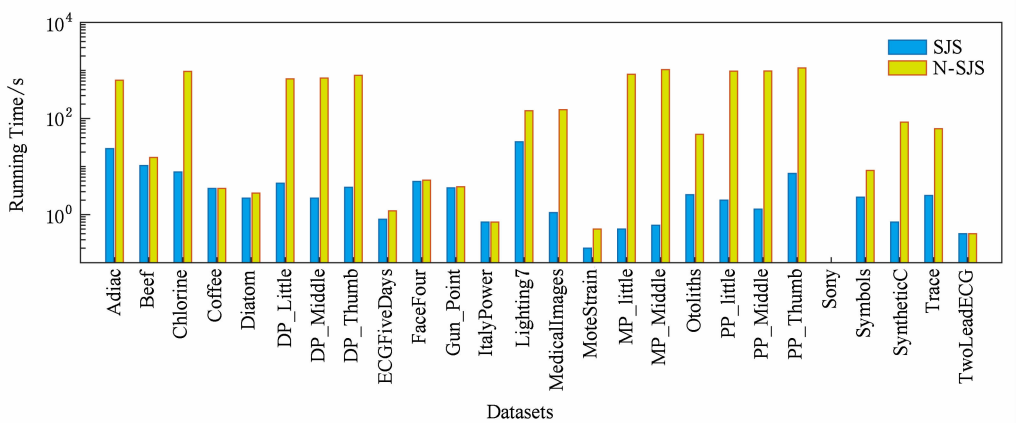


Fig. 8 Running time comparison of SJS and N-SJS algorithm

图 8 SJS 算法和 N-SJS 算法的运行时间对比

另一方面,算法 1 的主要目的是减少参与计算的时序数据的数量,加速 Shapelets 的提取,图 8 是两者的运行时间对比图(纵轴为对数坐标). 在绝大多数的数据集上,使用算法 1 之后,加速效果明显,如在 Adiac 数据集上的运行时间比为 23.6 s:626.1 s, Chlorine. 数据集上为 7.7 s:957.3 s. 仅有 4 个数据集没有加速效果,这是因为这些数据集的训练集本身非常小,且数据之间差异比较大,算法 1 将所有时序都作为关键数据. 但这并没有影响本文算法的效率,因为算法在这些数据集上使用全部时序数据的时间消耗非常小(Coffee:3.5 s, ItalyPower:0.7 s, Sony:0.1 s, TwoLeadECG:0.4 s).

5.3 实验结果对比

本节从分类准确率和运行时间 2 方面来验证本文算法 SJS 的性能.

5.3.1 分类准确率

表 2 列出了本文算法与所有对比算法在各个数据集上的分类准确率. 其中,每个数据集上的最优算法以加粗的形式表示,需要说明的是 IG, KW, FS 这 3 个方法在部分数据集上运行时间过长,无法在合理的运行时间内结束(超过 24 h),标记为"*". 表 2 的最后一行表示各个算法在不同数据集上具有最好分类效果的次数. 从该统计结果上来看, LTS 具有一定的优势,在 13 个数据集上取得最好的分类准确率,本文算法 SJS 仅次于 LTS,在 9 个数据集上处于领先,而标准的使用 IG 和 KW 作为度量标准的 Shapelets 算法效果最差.

然而,仅从算法在各个数据集上获胜的次数来评价算法性能是不全面的,为此,本文使用 5.1.3 节的方法来评价各个算法,其结果如图 9 所示. 从该关

Table 2 Testing Accuracy of Different Methods on the Datasets

表 2 不同方法在各个数据集上的测试准确率 %

Datasets	IG	KW	FS	FSH	IGSVM	LTS	FLAG	SJS
Adiac	29.9	26.6	15.6	57.5	23.5	51.9	**75.2**	72.8
Beef	50	33.3	56.7	50	90	76.7	83.3	**93.3**
Chlorine.	58.8	52	53.5	58.8	57.1	73	76	**87.3**
Coffee	96.4	85.7	**100**	92.9	**100**	**100**	**100**	**100**
Diatom.	76.5	62.1	76.5	87.3	93.1	94.2	96.4	97.7
DP_Little	*	*	*	60.6	66.6	**73.4**	68.3	68.1
DP_Middle	*	*	*	58.8	69.5	**74.1**	71.3	71.5
DP_Thumb	*	*	*	63.4	69.6	**75.2**	70.5	71.5
ECGFiveDays	77.5	87.2	99	99.8	99	**100**	92	98.5
FaceFour	84	44.3	75	92	**97.7**	94.3	90.9	**97.7**
Gun_Point	89.3	94	95.3	94	**100**	99.6	96.7	98
ItalyPower	89.2	91	93.1	91	93.7	**95.8**	94.6	**95.8**
Lighting7	49.3	48	41.1	65.2	63	**79**	76.7	75.3
MedicalImages	48.8	47.1	50.8	64.7	52.2	71.3	71.4	**74.8**
MoteStrain	82.5	84	84	83.8	88.7	**90**	88.8	89.8
MP_little	*	*	*	56.9	70.7	**74.3**	69.3	71.8
MP_Middle	*	*	*	60.3	76.9	**77.5**	75	75.5
Otoliths	67.1	60.8	57.9	60.8	64.1	59.4	64.1	**67.1**
PP_little	*	*	*	57.6	**72.1**	71	67.1	69.1
PP_Middle	*	*	*	61.6	**75.8**	74.9	73.8	74.7
PP_Thumb	*	*	*	55.7	**75.5**	70.5	67.4	70.7
Sony	85.7	72.7	**95.3**	68.6	92.7	91	92.9	91.2
Symbols	78.4	55.7	80.1	92.4	84.6	**94.5**	87.5	92.6
SyntheticC	94.3	90	95.7	94.7	87.3	97.3	**99.7**	**99.7**
Trace	98	94	**100**	**100**	98	**100**	99	98
TwoLeadECG	85.1	76.4	97	92.5	**100**	**100**	99	87.6
# Win	**0**	**0**	**3**	**1**	**7**	**13**	**3**	**9**

Note: The results highlighted in bold denote that the method gets the highest accuracy for this dataset and the "*" represents the corresponding method cannot finish in 24 hours.

键差异图可以看出,本文算法与最优的 LTS 算法差别很小(2.54:2.35),但与其他算法有比较明显的差别,如其他算法中最好的 FLAG 算法其平均次序已到 3.15.尽管 IGSVM 算法也在 7 个数据集上取得最好的准确率,但在平均次序上,与本文算法差距明显(2.54:3.58).

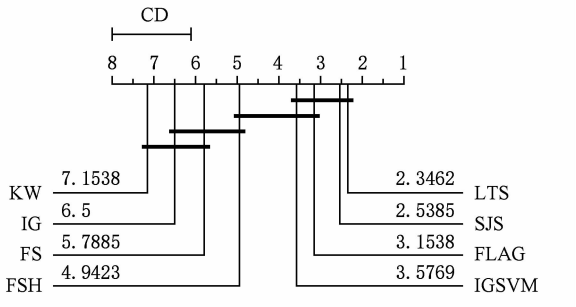


Fig. 9 Critical difference diagram for SJS and other baselines

图 9 SJS 和对比算法的关键差异图

综合上述 2 方面的分析,可以看出本文所提出的 SJS 算法具有很好的分类准确率,要优于目前绝大多数 Shapelets 提取算法.除此之外,SJS 的最大优势在于算法的稳定性,在几乎全部的数据集上,SJS 都保持很高准确率,而 LTS 算法则有些波动,如在 Adiac 和 Otoliths 数据集上,LTS 准确率不足 60%.更显著的是,当类别数增多时,SJS 优势明显,如在 37 类的 Adiac 数据集和 10 类的 MedicalImages 数据集上,SJS 都要优于 LTS.

5.3.2 运行时间比较

对于算法的评价,除了准确率之外,算法的运行时间也是一个重要的评价指标.对于实际应用,如金融、医疗等数据的处理,算法在可接受的时间内得到结果显得尤为重要.本文算法的核心在于时序数据间的相似性连接,子序列与 1 条时序数据的距离计算借助快速傅里叶变换完成(算法 2),其算法复杂度为 $O(m\text{lb}m)$(m 为时序数据长度),因此,计算 2 条时序数据相似性连接的时间复杂度为 $O((m-l+1)m\text{lb}m)$,即近似于 $O(m^2\text{lb}m)$.对于时序数据集,如果要处理所有时序数据对的相似性,算法复杂度会很高,但本文通过算法 1 提取每类中的关键数据,且相似性连接计算只在不同类的时序数据间进行,使得参与计算的时序条数大大减少,降低了运行时间消耗.

本文算法与其他对比算法的运行时间如表 3 所示(所有结果均在使用 Intel i7 CPU 和 16 GB 内存的计算机上得到),括号中的数字表示算法运行时间在每一个数据集上的排序次序.同表 2 一样,算法如果无法在合理的时间内完成,标记为"*",且次序用"(8)"表示.为了方便对比,本文按照式(6)计算每一算法的平均次序,表 3 中的最后一行 Average Rank 给出了相应的结果.该统计结果表明:FLAG 算法具有最好的时间效率(平均次序为 1.2),SJS 次之(平均次序为 2),标准使用 IG 作为度量的 Shapelets 提取算法的时间消耗最多.

Table 3 Running Time of Different Methods on the Datasets

表 3 不同方法在各个数据集上的运行时间

Datasets	Running Time/s(Rank)							
	IG	KW	FS	FSH	IGSVM	LTS	FLAG	SJS
Adiac	3 287(7)	1 349(5)	1 513(6)	288(3)	706(4)	80 596(8)	2.78(1)	23.6(2)
Beef	471(6)	484(7)	576(8)	154(3)	435(5)	414(4)	1.15(1)	10.5(2)
Chlorine.	9 751(8)	3 213(7)	3 050(6)	617(4)	1 181(5)	556(3)	6.88(1)	7.7(2)
Coffee	22.3(7)	21.8(5)	21.9(6)	16.5(4)	15.9(3)	76(8)	0.13(1)	3.5(2)
Diatom.	9.3(5)	8.99(3)	9.2(4)	13.7(7)	9.3(5)	88(8)	0.41(1)	2.2(2)
DP_Little	*(8)	*(8)	*(8)	401(3)	9516(5)	1 608(4)	1.81(1)	4.5(2)
DP_Middle	*(8)	*(8)	*(8)	456(3)	7 041(4)	12 528(5)	1.78(1)	2.2(2)
DP_Thumb	*(8)	*(8)	*(8)	392(3)	11 353(5)	2 073(4)	3.14(1)	3.7(2)
ECGFiveDays	19(8)	18.4(6)	18.6(7)	4.6(3)	11.7(5)	9.1(4)	0.08(1)	0.8(2)
FaceFour	1 021(8)	1 012(7)	1 010(6)	75(3)	410(5)	116(4)	0.26(1)	4.9(2)
Gun_Point	116.4(7)	112(6)	148.9(8)	7.6(3)	74.8(5)	14.1(4)	0.07(1)	3.6(2)
ItalyPower	0.38(5)	0.22(2)	0.22(2)	0.5(6)	0.22(2)	7.3(8)	0.03(1)	0.7(7)
Lighting7	3 442(7)	3 438(6)	3 584(8)	307(3)	1473(5)	965(4)	0.31(1)	32.8(2)
MedicalImages	4 347(8)	2 625(7)	2 616(6)	164(3)	1 547(4)	2 199(5)	1.69(2)	1.1(1)
MoteStrain	1.41(7)	1.29(4)	1.29(4)	1.4(6)	0.84(3)	6(8)	0.04(1)	0.1(2)

Continued (Table 3)

Datasets	Running Time/s(Rank)							
	IG	KW	FS	FSH	IGSVM	LTS	FLAG	SJS
MP_little	* (8)	* (8)	* (8)	460(3)	7 394(4)	10 779(5)	2.95(2)	0.5(1)
MP_Middle	* (8)	* (8)	* (8)	421(3)	12 102(5)	1 448(4)	2.23(2)	0.6(1)
Otoliths	17 864(7)	18 166(8)	17 789(6)	183(3)	8 536(5)	264(4)	1.25(1)	2.6(2)
PP_little	* (8)	* (8)	* (8)	393(3)	8 142(4)	9 899(5)	3.91(2)	2(1)
PP_Middle	* (8)	* (8)	* (8)	403(3)	4 753(4)	6 819(5)	2.19(2)	1.3(1)
PP_Thumb	* (8)	* (8)	* (8)	416(3)	8 209(4)	13 675(5)	4.2(1)	7.2(2)
Sony	1.17(7)	1.02(4)	1.02(4)	1.1(6)	0.75(3)	25.4(8)	0.04(1)	0.1(2)
Symbols	3 325(7)	3 318(6)	3 622(8)	59.4(3)	1 263(5)	528(4)	0.85(1)	2.3(2)
SyntheticC	291(6)	164(5)	161(4)	39.4(3)	922(8)	293(7)	0.26(1)	0.7(2)
Trace	11 542(7)	11 622(8)	11 411(6)	98.7(3)	4 838(5)	394(4)	0.35(1)	2.5(2)
TwoLeadECG	0.48(6)	0.42(4)	0.42(4)	1.1(7)	0.26(2)	5.2(8)	0.08(1)	0.4(3)
Average Rank	**7.2**	**6.3**	**6.4**	**3.7**	**4.5**	**5.3**	**1.2**	**2.0**

Note: The results highlighted in bold denote that the method gets the highest accuracy for this dataset and the "*" represents the corresponding method cannot finish in 24 hours.

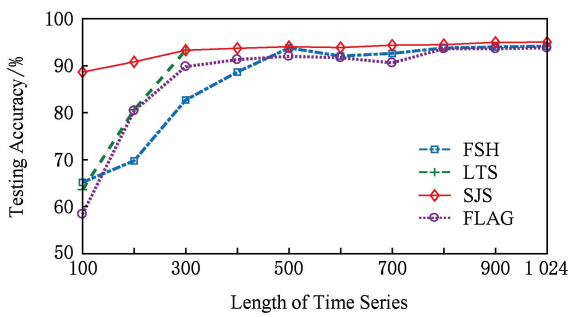
虽然运行时间的平均次序能够体现出算法的时间效率,但没有体现具体运行时间差异的大小. 从表 3 中可以看出,尽管 FLAG 要优于 SJS,但在大部分数据集上,两者几乎在同一个数量级范围内,均在几秒甚至更短的时间内得出结果. 相反,其他算法则需要很长的时间,特别是准确率最高的 LTS 算法,稍大一点的数据集都需要几千甚至上万秒的时间消耗.

本节的实验表明:LTS 具有最好的分类效果,但运行时间长,尤其在类别数多时,时间消耗大,如在 Adiac 数据集上,差不多需要 24 h 的时间;FLAG 的时间效率最好,但分类准确率不高,而本文算法 SJS 不仅在分类准确率上接近 LTS 算法,而且在运行时间上与最优的 FLAG 具有同等的数量级. 因此,综合分类准确率和运行时间 2 方面,SJS 的实用性更好.

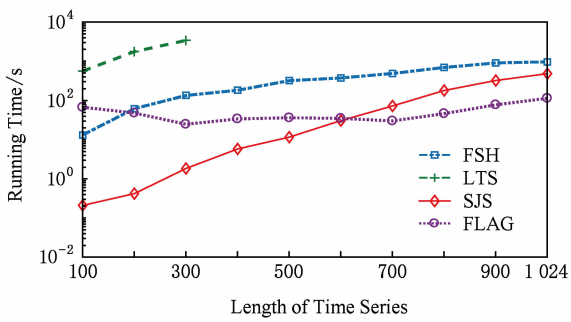
5.4 算法可扩展性

本文以 UCR archives 中的最大时序数据集之一 StarLightCurves 来验证算法的可扩展性. 该数据集包含 3 类的"星光"数据,共计 9 236 条,其中 Eclipsed Binaries 有 2 580 条,Cepheids 有 1 329 条,RR Lyrae Variables 有 5 236 条. 实验中采用默认的训练集和数据集划分. Shapelets 的提取时间主要受 2 方面的影响:1)时序的长度;2)训练集的大小. 本文从这两方面进行实验. 由于对比算法 IG,KW,FS,IGSVM 在 StarLightCurves 数据集上的运行时间太长,难以获得实验结果,因此,本文舍弃与这些算法的比较.

首先验证时序长度的变化对于分类准确率和运行时间的影响,具体设置为:固定训练集中时序的数量,将时序的长度从 100 变化至 1 024,以 100 为间隔单元. 实验中,LTS 算法受限于时间和空间消耗,在长度大于 300 时难以得到结果,因而只有 3 个实验数据. 实验结果如图 10 所示,图 10(a)是随着时序长度的增加,各对比算法在测试集上分类准确率的变化,图 10(b)是对应运行时间的变化. 从图 10(a)中可以看出,本文算法 SJS 受时序数据长度变化



(a) Testing accuracy for varying the length of time series



(b) Running time for varying the length of time series

Fig. 10 Experimental results for varying the length of time series

图 10 时序数据长度变化时的实验结果

影响最小且相对稳定,而其他算法在长度为 100 和 200 时,分类准确率很低且在长度增大过程中分类结果有波动,这说明,SJS 在提取 Shapelets 过程中能够很好的发现当前处理的数据集中最具类别可分性的子序列. 运行时间对比的结果与表 3 基本一致,即 FLAG 具有最优的运行时间,SJS 次之,但长度在 100～600 之间时,SJS 要优于 FLAG,这充分验证了 SJS 的主要时间消耗来自时序数据间的相似性连接.

其次,固定时序数据的长度为 1 024,将训练集中时序数据数量从 100 增加至 1 000,使用默认的测试集. 由于 LTS 算法无法在合理的运行时间内得到结果,所以本实验中仅有 SJS,FSH,FLAG 这 3 个算法的结果,如图 11 所示. 与 FSH 和 FLAG 在训练集增大时,分类准确率有起伏不同,SJS 一直处于上升的趋势,如图 11(a),说明当训练集增大时,FSH 和 FLAG 所提取的 Shapelets 可能会不同,而 SJS 则会在原有提取出的 Shapelets 基础上,对新增加的训练集数据提取 Shapelets,进而提高分类准确率. 这一特性对于优化算法的运行效率更为重要,如图 11(b)所示,尽管在同等规模的数据集上 FLAG 要优于 SJS(如图 11(b)所示),但 SJS 在应对数据集变化时可以重用之前结果,能够减少后续计算量,如图 11(b)中的 SJS-R 曲线,因此,SJS 有更好的增量扩展性.

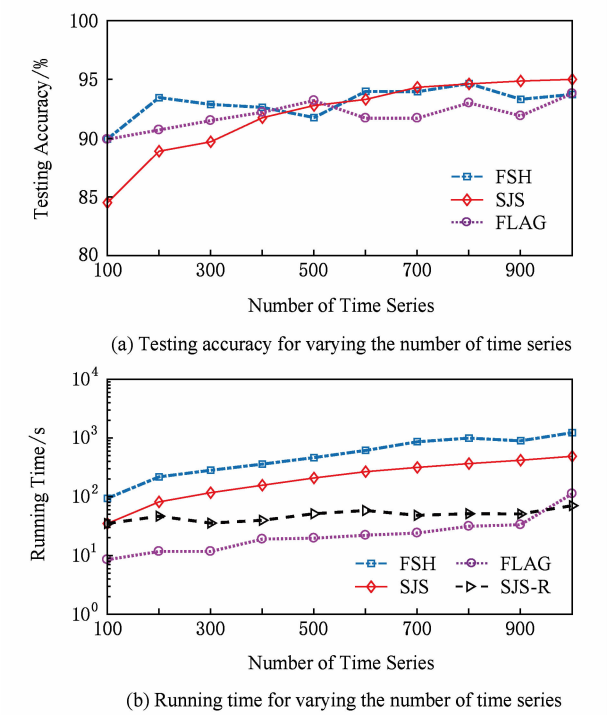


Fig. 11 Experimental results for varying the number of training time series

图 11 训练集时序数据数量变化时的实验结果

6 结 论

本文针对时序数据中的 Shapelets 提取问题进行研究,提出了基于相似性连接策略的 Shapelets 提取算法 SJS. 该方法以子序列为基本单元描述时序数据之间的相似性,通过不同类别的时序数据之间的差异向量构建候选矩阵,然后提取候选矩阵中大差异值对应的子序列构成 Shapelets. 在大量真实时序数据集上进行分类的实验表明,本文方法所提取的 Shapelets 不仅能获得良好的分类准确率,还具有很高的时间效率,实用性强. 同时,可扩展性实验表明:在训练数据集增大时,本文算法能够增量提取 Shapelets,可以减少大量重复计算.

参 考 文 献

[1] Su Weixing, Zhu Yunlong, Liu Fang, et al. Outliers and change-points detection algorithm for time series [J]. Journal of Computer Research and Development, 2014, 51(4): 781-788 (in Chinese)
(苏卫星, 朱云龙, 刘芳, 等. 时间序列异常点及突变点的检测算法[J]. 计算机研究与发展, 2014, 51(4): 781-788)

[2] Wu Honghua, Liu Guohua, Wang Wei. Similarity matching for uncertain time series [J]. Journal of Computer Research and Development, 2014, 51(8): 1802-1810 (in Chinese)
(吴红花, 刘国华, 王伟. 不确定时间序列的相似性匹配问题[J]. 计算机研究与发展, 2014, 51(8): 1802-1810)

[3] Esling P, Agon C. Time-series data mining [J]. ACM Computing Surveys (CSUR), 2012, 45(1): 1-34

[4] Ding Hui, Trajcevski G, Scheuermann P, et al. Querying and mining of time series data: Experimental comparison of representations and distance measures [J]. Proc of the VLDB Endowment, 2008, 1(2): 1542-1552

[5] Lines J, Bagnall A. Time series classification with ensembles of elastic distance measures [J]. Data Mining and Knowledge Discovery, 2015, 29(3): 565-592

[6] Ratanamahatana C A, Keogh E. Three myths about dynamic time warping data mining [C] //Proc of the 5th SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2005: 506-510

[7] Chen Lei, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories [C] //Proc of the 24th ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2005: 491-502

[8] Keogh E J, Pazzani M J. Derivative dynamic time warping [C] //Proc of the 1st SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2001: 1-11

[9] Jeong Y S, Jeong M K, Omitaomu O A. Weighted dynamic time warping for time series classification [J]. Pattern Recognition, 2011, 44(9): 2231-2240

[10] Marteau P F. Time warp edit distance with stiffness adjustment for time series matching [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009, 31(2): 306-318

[11] Stefan A, Vassilis A, Gautam D. The move-split-merge metric for time series [J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(6): 1425-1438

[12] Batista G E, Wang Xiaoyue, Keogh E J. A complexity-invariant distance measure for time series [C] //Proc of the 11th SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2011: 699-710

[13] Ye Lexiang, Keogh E. Time series shapelets: A novel technique that allows accurate, interpretable and fast classification [J]. Data Mining and Knowledge Discovery, 2011, 22(1): 149-182

[14] Mueen A, Keogh E, Young N. Logical-shapelets: An expressive primitive for time series classification [C] //Proc of the 17th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2011: 1154-1162

[15] Lin J, Keogh E, Li Wei, et al. Experiencing SAX: A novel symbolic representation of time series [J]. Data Mining and Knowledge Discovery, 2007, 15(2): 107-144

[16] Rakthanmanon T, Keogh E. Fast shapelets: A scalable algorithm for discovering time series shapelets [C] //Proc of the 13th SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2013: 668-676

[17] Hills J, Lines J, Baranauskas E, et al. Classification of time series by shapelet transformation [J]. Data Mining and Knowledge Discovery, 2014, 28(4): 851-881

[18] Lines J, Davis L M, Hills J, et al. A shapelet transform for time series classification [C] //Proc of the 18th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2012: 289-297

[19] Yeh C C M, Zhu Yan, Ulanova L, et al. Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets [C] //Proc of the 16th Int Conf on Data Mining (ICDM). Piscataway, NJ: IEEE, 2016: 1317-1322

[20] Yuan Jidong, Wang Zhihai, Han Meng, et al. A logical shapelets transform for time series classification [J]. Chinese Journal of Computers, 2015, 38(7): 1448-1459 (in Chinese) (原继东, 王志海, 韩萌, 等. 基于逻辑 shapelets 转换的时间序列分类算法[J]. 计算机学报, 2015, 38(7): 1448-1459)

[21] Chang Kaiwei, Deka B, Hwu W M W, et al. Efficient pattern-based time series classification on GPU [C] //Proc of the 12th Int Conf on Data Mining (ICDM). Piscataway, NJ: IEEE, 2012: 131-140

[22] Wistuba M, Grabocka J, Schmidt-Thieme L. Ultra-fast shapelets for time series classification [EB/OL]. 2015 [2015-03-17]. https://arxiv.org/abs/1503.05018

[23] Zhang Zhenguo, Zhang Haiwei, Wen Yanlong, et al. Accelerating time series shapelets discovery with key points [G] //LNCS 9932: Proc of the 12th Asia-Pacific Web Conf. Berlin: Springer, 2016: 330-342

[24] Grabocka J, Schilling N, Wistuba M, et al. Learning time-series shapelets [C] //Proc of the 20th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2014: 392-401

[25] Hou Lu, Kwok J T, Zurada J M. Efficient learning of timeseries shapelets [C] //Proc of the 30th AAAI Conf on Artificial Intelligence. Menlo Park, CA: AAAI, 2016: 1209-1215

[26] Bagnall A, Lines J, Bostrom A, et al. The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances [J]. Data Mining and Knowledge Discovery, 2017, 31(3): 606-660

[27] Rakthanmanon T, Campana B, Mueen A, et al. Searching and mining trillions of time series subsequences under dynamic time warping [C] //Proc of the 18th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2012: 262-270

[28] Begum N, Keogh E. Rare time series motif discovery from unbounded streams [J]. Proc of the VLDB Endowment, 2014, 8(2): 149-160

[29] Mueen A, Zhu Yan, Yeh M et al. The fastest similarity search algorithm for time series subsequences under Euclidean distance [EB/OL]. [2017-08-12]. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

[30] Bagnall A, Davis L, Hills J, et al. Transformation based ensembles for time series classification [C] //Proc of the 12th SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2012: 307-318

[31] Demšar J. Statistical comparisons of classifiers over multiple data sets [J]. Journal of Machine Learning Research, 2006, 7(1): 1-30

Zhang Zhenguo, born in 1981. PhD. His main research interests include machine learning and data mining.

Wang Chao, born in 1990. Master. His main research interests include information retrieval and data mining.

Wen Yanlong, born in 1979. PhD. Senior engineer. His main research interests include database, data mining and information retrieval.

Yuan Xiaojie, born in 1963. PhD. Professor, PhD supervisor. Her main research interests include data management and data mining.