软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(9):2311−2325 [doi: 10.13328/j.cnki.jos.004702]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Shapelet 剪枝和覆盖的时间序列分类算法*

原继东[1,2], 王志海[1,2], 韩 萌[1,2]

[1](北京交通大学 计算机与信息技术学院,北京 100044)

[2](交通数据分析与挖掘北京市重点实验室(北京交通大学),北京 100044)

通讯作者: 王志海, E-mail: 12112078@bjtu.edu.cn

摘 要: 时间序列 shapelets 是时间序列中能够最大限度地表示一个类别的子序列.解决时间序列分类问题的有效途径之一是通过 shapelets 转换技术,将 shapelets 的发现与分类器的构建相分离,其主要优点是优化了 shapelets 的选择过程,并能够灵活应用不同的分类策略.但该方法也存在不足:一是在 shapelets 转换时,用于产生最好分类结果的 shapelets 数量是很难确定的;二是被选择的 shapelets 之间往往存在着较大的相似性.针对这两个问题,首先提出了一种简单有效的 shapelet 剪枝技术,用于过滤掉相似的 shapelets;其次,提出了一种基于 shapelets 覆盖的方法来确定用于数据转换的 shapelets 的数量.通过在多个数据集上的测试实验,表明了所提出的算法具有更高的分类准确率.

关键词: 时间序列分类;shapelet 剪枝;shapelet 覆盖

中图法分类号: TP311

中文引用格式: 原继东,王志海,韩萌.基于 Shapelet 剪枝和覆盖的时间序列分类算法.软件学报,2015,26(9):2311−2325. http://www.jos.org.cn/1000-9825/4702.htm

英文引用格式: Yuan JD, Wang ZH, Han M. Shapelet pruning and shapelet coverage for time series classification. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2311−2325 (in Chinese). http://www.jos.org.cn/1000-9825/4702.htm

Shapelet Pruning and Shapelet Coverage for Time Series Classification

YUAN Ji-Dong[1,2], WANG Zhi-Hai[1,2], HAN Meng[1,2]

[1](School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

[2](Beijing Key Laboratory of Traffic Data Analysis and Mining (Beijing Jiaotong University), Beijing 100044, China)

Abstract: Time series shapelets are subsequences of time series that can maximally represent a class. One of the most promising approaches to solve the problem of time series classification is to separate the process of finding shapelets from classification algorithm by adopting a shapelet transformation. The main advantages of that technique are that it optimizes the process of shapelets selection and different classification strategies could be applied. Important limitations also exist in that method. First, although the number of shapelets selected for the transformation directly affects the classification result, the quantity of shapelets which yields the best data for classification is hard to be decided. Second, previous algorithms often inevitably result in similar shapelets among the selected shapelets. This work addresses the latter problem by introducing an efficient and effective shapelet pruning technique to filter similar shapelets and decrease the number of candidate shapelets at the same time. On this basis, a shapelet coverage method is proposed for selecting the number of shapelets for a given dataset. Experiments using the classic benchmark datasets for time series classification demonstrate that the proposed transformation can improve classification accuracy.

Key words: time series classification; shapelet pruning; shapelet coverage

在时间序列分类问题中,我们将任意实值型有次序的数据看作一条时间序列[1].它与传统分类问题之间的差别在于:传统分类数据的属性次序是不重要的,并且变量之间的相互关系独立于它们的相对位置;而对于时间

* 基金项目: 北京市自然科学基金(4142042); 中央高校基本科研基金(2015YJS049)
收稿时间: 2014-03-07; 修改时间: 2014-05-15; 定稿时间: 2014-07-30

序列数据而言,在寻找最佳的辨别性特征时,变量的次序起着至关重要的作用.在过去的 10 多年中,针对时间序列分类问题的研究主要集中在基于不同距离度量方式的最近邻(1-NN)算法上,如基于欧式距离或者动态时间规整(dynamic time wrapping,简称 DTW)的 1-NN 等[2−4].近年来,基于时间序列 shapelets 的分类方法引起了研究者广泛的关注[1,5−8].

时间序列 shapelets 是时间序列中能够最大限度地表示一个类别的子序列[5,6].基于时间序列 shapelets 的分类方法具有分类准确性高、分类速度快、可解释性强等优点,它的具体概念是由 Ye 等人[5]所提出的.图 1 所示为解决经典 *Gun*/*NoGun* 问题[5,9]的最佳 shapelet,该 shapelet 准确地展现了 *Gun* 与 *NoGun* 序列间的不同.如 Esling 等人[10]所述,此方法大大缩小了时间序列和形状分析间的差距.

Fig.1 Best shapelet

图 1 Shapelet 示例

基于 shapelets 的时间序列分类算法可分为两类:

- 第 1 类方法在发现时间序列 shapelets 的同时构造分类器.在文献[5]中,作者采用信息增益度量每一条时间序列的候选 shapelet,并选取当前信息增益最大的 shapelet 作为决策树的分裂节点.该方法通过不断地递归搜索当前最好的 shapelets 完成决策树的构建.Mueen 等人[7]针对 shapelets 在可表示性方面的不足,提出了一种根据 shapelets 的合取或析取构建决策树的算法,并采用了多种加速技术来加快分类器的构建.Rakthanmanon 等人[8]针对前两种 shapelets 发现算法在时间效率上的不足,提出了一种基于 SAX(symbolic aggregate approximation)表示的快速 shapelets 发现算法,尽管此算法发现的是近似的而不是精确的 shapelets,但它在提升 shapelets 发现效率的同时,保持了一定的分类准确性.
- 第 2 类方法将时间序列 shapelets 的选择过程当作单独的预处理步骤来处理,即:通过 shapelets 转换,将 shapelets 的发现和分类器的构造相分离.其主要优点是优化了 shapelets 的选择过程,并能够灵活应用不同的分类策略.Bagnall 等人[11]指出:解决时间序列分类问题的最简单方式,是将时间序列映射到其他辨别性特征容易被检测到的空间.作者通过在建造分类器之前将时间序列分类问题转换到其他空间,极大地提升了分类的准确性.Lines 等人[1]首次提出了基于 shapelets 转换的时间序列分类方法,此方法通过在构建分类器之前创建新的分类数据,使得它在保持 shapelets 的解释力的同时提升了分类的准确性.另外,时间序列 shapelets 还被应用于其他方面,比如聚类[12]、早期分类[13]、姿势识别[14]、天气预测[15]等等.

然而,基于 shapelets 转换的时间序列分类方法存在两个主要问题:

- 一是 shapelets 转换时,所选取的时间序列 shapelets 的数量影响着分类结果的好坏,而用于产生最好分类结果的 shapelets 数量是很难确定的;
- 二是被选择的 shapelets 之间往往存在着较大的相似性.

例如,对于经典的 *Gun*/*NoGun* 问题,通过文献[1]中所提出的 ShapeletFilter 算法而得到的前 10 个 shapelets 如图 2(a)所示.从图中可以观察到:这 10 个 shapelets 聚成了两簇,即,shapelets 之间存在着极大的相似性.我们只需要找出这两簇中最具有辨别性的 2 个 shapelets(如图 2(b)所示)就可以涵盖这 10 个 shapelets,从而允许其他有辨别性的 shapelets 参与到数据转换中,并有利于提高分类准确率.

对于数据转换时所需 shapelets 的数量,我们借鉴属性选择中的序列覆盖思想[16],提出了 shapelet 覆盖的概念,并根据 shapelets 对实例的覆盖情况来选取 shapelets.

(a) *Gun*/*NoGun* 数据集中最好的 10 个 shapelets (b) *Gun*/*NoGun* 数据集中最具有辨别性的 2 个 shapelets

Fig.2 Similar shapelets and discriminative shapelets

图 2 相似 shapelets 与辨别性 shapelets

本文的主要贡献有:

(1) 提出了一种简单有效的 shapelet 剪枝方法,它能够过滤相似的 shapelets,并减少候选 shapelets 样本的数量;

(2) 提出了一种基于 shapelets 覆盖的方法来确定用于数据转换的 shapelets 的数量,保证了 shapelets 对实例的覆盖;

(3) 将所提出的算法和其他基于 shapelets 的时间序列分类算法以及基于 DTW 和欧式距离的 1-NN 分类器作对比,阐明了我们所提出算法的分类准确性.

本文第 1 节将阐述时间序列分类问题的相关定义和 shapelets 转换方法的研究背景.第 2 节描述本文所提出的 shapelet 选择和 shapelet 覆盖等方法.第 3 节给出实验描述并评估所提出的算法.第 4 节进行相应的总结以及对进一步工作的展望.

1 定义与背景

1.1 定义与符号

一条时间序列是一组序列数据,它常常是在相等间隔的时间段内,依照给定的采样率,对某潜在的过程进行观察的结果.时间序列分类的目标是:首先,从标定类标的训练集中学习到能够区分不同类别的有鉴别性的特征;然后,当一条未标定的时间序列到来时,它能够自动决定该时间序列的类标.假设一个时间序列数据集有 n 条时间序列:$D=\{T_1,T_2,\ldots,T_n\}$,每一条时间序列有 m 个观测值和一个类值 c_i,即 $T_i=\langle t_{i1},t_{i2},\ldots,t_{im},c_i\rangle$.我们的目标是,构造一个将时间序列的观测值映射到类值的函数.为方便起见,假设数据集 D 中每一条时间序列具有相同数目的观测值.文中涉及的符号见表 1.

Table 1 Symbols

表 1 符号表

符号	释义
D	时间序列数据集
T, X, Y	时间序列
S, s	时间序列的子序列
$\|x\|, l$	时间序列子序列的长度
$dist(x,y)$	时间序列 X 与 Y 之间的规范化距离
$subdist(x,y)$	子序列距离
$\bar{X},\sigma_x$	均值与方差
$N, \|D\|$	数据集中时间序列的条数
τ	距离阈值
(s,τ)	一个 shapelet
E	熵
I	信息增益
G	分裂间隔
L	orderline

定义 1(时间序列及其子序列). 时间序列 T 是一条实值的序列,可表示为 $t_1,t_2,\ldots,t_m$.一条时间序列的子序列 $S_{i,l}=t_i,t_{i+1},\ldots,t_{i+l-1}$ 是一条 T 中从位置 i 开始长度为 l 的连续的子序列,这里,$1\leqslant i\leqslant m-l+1$.注意:数据点 $t_1,t_2,\ldots,t_m$ 是

按照时间顺序排列的,两两之间具有相同的时间间隔.时间序列的子序列可以当作是时间序列的局部特征.

定义 2(时间序列间的距离). 所有的分类问题都依赖于数据间的相似性度量,时间序列分类问题也不例外.对于给定的具有相同长度 m 的两条时间序列 X 和 Y,我们可以使用长度规范化的欧式距离来度量他们之间的不同,见公式(1).

$$dist(x,y)=\sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i-y_i)^2} \tag{1}$$

注意:为获得两个时间序列之间距离的有效比对并保证缩放和偏移的不变性,在计算真正的距离之前,我们必须使用 z-规范化方法(见公式(2))对每个时间序列进行规范化处理[3,7].

$$x_{norm}=\frac{x-\bar{X}}{\sigma_X} \tag{2}$$

许多时间序列数据挖掘算法只需要对比相等长度的时间序列.而时间序列 shapelets 需要计算一条较短的时间序列子序列与一条较长的时间序列间的距离.为得到此结果,短的时间序列须在长的序列上滑动,以得到它们之间的最短距离,我们称为子序列距离,并定义为

$$subdist(x,y)=\min(dist(x,y_{|x|})) \tag{3}$$

其中,$y_{|x|}$表示时间序列 Y 中长度为$|x|$的子序列.需要注意的是,$subdist()$得到的是两条时间序列间的最小距离.为减少计算规范化距离的时间复杂度和重复利用中间结果,在本文中采用了充分统计量(sufficient statistics)技术来加速此计算过程.相比较于现有的方法,充分统计量技术能够将发现 shapelets 的时间复杂度降低一个数量级,请读者参照文献[7]中的描述,以进一步了解充分统计量技术.

定义 3(shapelets 与信息增益). shapelet 是由 D 中某一实例的子序列 s 和分裂阈值τ所组成的元组(s,τ).τ表示距离的阈值.shapelet 会将数据集 D 分割成两个不相交的子集,其中,

$$D_{left}=\{x:x\in D,subdist(s,x)\leqslant\tau\},D_{right}=\{x:x\in D,subdist(s,x)>\tau\}.$$

本文采用信息增益度量 shapelets 的鉴别能力,令 $N_1=|D_{left}|,N_2=|D_{right}|$,一个 shapelet 的信息增益为

$$I(s,\tau)=E(D)-\frac{N_1}{N}E(D_{left})-\frac{N_2}{N}E(D_{right}) \tag{4}$$

其中,$E(D)$为数据集 D 的信息熵,N 为数据集 D 中时间序列的条数.

定义 4(分裂间隔). shapelet 的分裂间隔为

$$G(s,\tau)=\frac{1}{N_2}\sum_{x\in D_{right}}subdist(s,x)-\frac{1}{N_1}\sum_{x\in D_{left}}subdist(s,x) \tag{5}$$

即为分裂点右侧实例的平均距离减去分裂点左侧实例的平均距离.

定义 5(orderline). orderline L 是数组的一维表示形式,它按照递增的顺序记录了候选 shapelet 与数据集 D 中所有时间序列间的距离.

我们将用图 3 所示的例子来阐释 shapelet 和 orderline 的概念.左边方框中是一个候选的 shapelet,右边的 orderline 记录了该候选 shapelet 与数据集 D 中每个时间序列之间的距离,τ为分裂点的阈值.

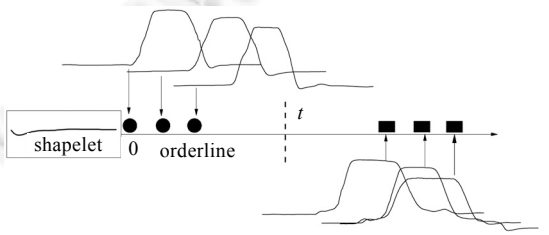

Fig.3 An example of orderline

图 3 Orderline 示例

在图 3 中:数据集 D 为经典的 *Gun*/*NoGun* 数据集,圆形表示 *NoGun* 序列与候选 shapelet 之间的距离,它们

与候选 shapelet 间的距离较小;矩形表示 *Gun* 序列与候选 shapelet 之间的距离,它们与候选 shapelet 间的距离较大.orderline 创建之后,我们就可以计算分裂点、分裂间隔以及候选的 shapelet 的信息增益等.最终选取最优的一个或多个 shapelets 进行分类.

1.2 Shapelets转换技术

为阐述我们的贡献,我们将首先描述文献[1]中提出的基于 shapelets 转换的时间序列分类方法.此方法的主要动机是:将 shapelets 的发现与分类器的构建相分离,从而允许转换后的数据应用于传统分类器中.相应的算法实现主要包括 3 个步骤:首先,通过扫描一次数据集,找到最好的 k 个 shapelets;其次,使用交叉验证方法估计能够得到最好分类结果的 shapelets 的个数 k;然后,将每一条时间序列转换成具有 k 个属性的实例.即,让每个属性代表一个 shapelet,属性值为 shapelet 与该时间序列之间的距离.这样就创建了一个新的数据集;最后,将转换后的数据集用于朴素贝叶斯、贝叶斯网络、决策树、1-NN 等分类器进行分类.

具体的算法描述见算法 1.

算法 1. *ShapeletFilter*(T,min,max,k).

输入:时间序列 T、最小长度 min、最大长度 max、shapelet 个数 k;

输出:k 个最好的 shapelets *kShapelets*.

```
1:  kShapelets←∅;
2:  for i←0 to |T| do {T 中的每一条时间序列}
3:      shapelets←∅;
4:      for l←min to max do {T_i 中的每一个长度}
5:          for u←0 to |T_i|−l+1 do {每一个起始位置}
6:              S←T_i(u,l);
7:              for m←0 to |T| do {候选 shapelet S 与每一条时间序列的距离}
8:                  D_s←subdist(S,T_m);
9:              orderline←sort(D_s); {创建 orderline}
10:             quality←assessCandidate(S,orderline,D_s); {计算 shapelets 最大信息增益}
11:             shapelets.add(S,quality);
12:     sortByQuality(shapelets); {根据信息增益大小排序}
13:     removeSelfSimilar(shapelets); {移除自相似的 shapelets}
14:     kShapelets←merge(k,kShapelets,shapelets); {保留当前最好的 k 个 shapelets}
15: return kShapelets;
```

算法 1 描述了从数据集中抽取 k 个最好的 shapelets 的过程.根据最大最小长度的设置范围,每一条时间序列的每一种可能长度的子序列都被做了相应的评估,并最终得到最好的 k 个 shapelets.得到一个候选的 shapelet 之后,需要计算该 shapelet 与每一条时间序列间的距离,构建 orderline(第 2 行~第 9 行),并计算能得到最大信息增益的分裂点(第 9 行~第 11 行).得到所有候选 shapelets 之后,根据其信息增益大小进行排序(第 12 行).若两个 shapelets 来自同一条时间序列并有重叠之处,则认为它们是自相似的,此时需要保留较好的 shapelet,并移除自相似的 shapelet(第 13 行).一旦得到一条时间序列中所有非自相似的 shapelets,就将它们与现有的最好的 k 个 shapelets 相结合,并保留最好的 k 个(第 14 行).

得到 k 个最好的 shapelets 之后,利用它们将原有的时间序列转换成新的数据集的方法见算法 2.数据转换时使用的子序列距离计算方法如算法 3 所示(第 5 行).shapelets 与时间序列比对时,都进行了规范化处理(算法 3 第 2 行、第 5 行).对于 D 中的每一条时间序列 D_i,它的子序列距离通过与 $S_j(j=0,...,k-1)$计算得出.得到的 k 个距离用于构建转换后数据集,即:新的数据集中,每一条实例有 k 个属性,每一个属性值都对应于相应的 shapelet 到原始时间序列的距离(第 6 行、第 7 行).

算法 2. *TransformData*(*S*,*D*).

输入:最好的 *k* 个 shapelets *S*、数据集 *D*;

输出:转换后的数据集 *output*.

1: *output*←∅;
2: **for** *i*←0 to |*D*| **do** {*D* 中的每一条时间序列}
3: *transformed*←∅;
4: **for** *j*←0 to |*S*| **do** {*S* 中的每一个 shapelet}
5: *dist*←***subdist***(S_j,D_i); {子序列距离}
6: *transformed.add*(*dist*);
7: *output.add*(*transformed*);
8: **return** *output*;

算法 3. *subdist*(*x*,*y*).

输入:时间序列 *x* 和 *y*,|*x*|≤|*y*|;

输出:*x* 与 *y* 的规范化距离.

1: *bestSum*←MAX_*VALUE*;
2: *x*←*zNorm*(*x*); {规范化}
3: **for** *i*←0 to |*y*|−|*x*|+1 **do** {*y* 中的每一个起始位置}
4: *sum*←0;
5: *z*←*zNorm*($y_{i,|x|}$); {规范化}
6: **for** *j*←0 to |*x*| **do** {计算欧氏距离}
7: *sum*←*sum*+$(z_i-x_i)^2$;
8: *bestSum*←min(*bestSum*,*sum*); {取最小值}
9: **return** $(bestSum/|x|)^{1/2}$;

2 辨别性 Shapelets 转换技术

本文提出的转换算法通过 4 个主要步骤处理 shapelets:

- 首先,通过扫描一次数据集,找到所有候选的 shapelets;
- 其次,通过使用本文提出的 shapelet 剪枝方法,从候选 shapelets 中移除相似的 shapelets;
- 再次,使用本文新提出的 shapelet 覆盖方法来确定用于数据转换的 shapelets 的数目 *k*;
- 最后,将每条时间序列转换成具有 *k* 个属性的实例,并将多种不同的分类策略应用到时间序列分类中.

2.1 候选Shapelets

生成所有候选 shapelets 的过程见算法 4.此算法处理数据的过程与文献[1]中的 shapelet 生成算法相类似.主要区别在于:我们算法的目标是找到所有的候选 shapelets 而不是最好的 *k* 个.原因是:任意时间序列的任意长度的子序列都有可能是辨别性 shapelet,所以直接省略了参数 *k* 的设置.需要注意的是:当自相似的 shapelets 从一条时间序列中移除之后(第 13 行),将它们添加到候选的 shapelets 中,并按照它们信息增益的降序排列.此过程不断迭代,直至处理完所有的时间序列(第 14 行).

算法 4. *CandidateShapelets*(*T*,min,max).

输入:时间序列 *T*、最小长度 min、最大长度 max;

输出:候选 shapelets *CandidateShapelets*.

1: *CandidateShapelets*←∅;
2: **for** *i*←0 to |*T*| **do** {*T* 中的每一条时间序列}
3: *shapelets*←∅;

```
4:    for l←min to max do {Ti 中的每一个可能长度}
5:      for u←0 to |Ti|–l+1 do {每一个起始位置}
6:        S←Ti(u,l);
7:        for m←0 to |T| do {计算候选 shapelet S 与每一条时间序列的距离}
8:          Ds←subdist(S,Tm);
9:        orderline←sort(Ds); {创建 orderline}
10:       quality←assessCandidate(S,orderline,Ds); {计算 shapelets 最大信息增益}
11:       shapelets.add(S,quality);
12:   sortByQuality(shapelets); {根据信息增益大小排序}
13:   removeSelfSimilar(shapelets); {移除自相似的 shapelets}
14:   CandidateShapelets.add(shapelets); {保留所有候选的 shapelets}
15: return CandidateShapelets;
```

2.2 Shapelet剪枝

我们提出 shapelet 剪枝方法的主要目的是:剪去相似的 shapelets,从而允许更多有辨别性的 shapelets 参与到数据转换中.为了更好地描述 shapelet 剪枝方法,我们将首先描述相似 shapelets 的概念.

定义 6(相似 shapelets). 对于两个 shapelets,(S_1,τ_1)和(S_2,τ_2),(S_1,τ_1)优于(S_2,τ_2).若它们所在时间序列的类标相同,并且两者间的子序列距离 $subdist(S_1,S_2)$小于(S_1,τ_1)的分裂阈值τ_1,那么两者为相似的 shapelets.

需要注意的是:定义的相似 shapelets 主要强调了形状的相似性,作比较的两个 shapelets 往往来自不同的时间序列;而自相似 shapelets 存在于同一条时间序列,并具有重叠的部分.由于τ_1是能够获取最大信息增益的距离阈值,当 $subdist(S_1,S_2)<\tau_1$ 时,表示 S_1 和 S_2 具有相似的形状,并且 S_1 能够在大部分情况下替代 S_2.例如,从图 2(a)中可以观察到,解决 *Gun*/*NoGun* 问题的前 10 个 shapelets 间存在着极大的形状相似性.Shapelet 剪枝之后,我们剪去了相似的 shapelets,并能够使用 2 个不相似的 shapelets 来代表最好的 10 个(如图 2(b)所示).过滤相似 shapelets 的过程见算法 5.

算法 5. *ShapeletsPrune*(*candidateShapelets*).

输入:候选 shapelets *candidateShapelets*;

输出:不相似的 shapelets *NoSimilarShapelets*.

```
1:  NoSimilarShapelets←∅;
2:  size←candidateShapelets.size(); {候选 shapelets 的个数}
3:  for i←0 to size do {每一个候选 shapelet}
4:    selectShapelet←candidateShapelets.get(i); {获取候选 shapelet}
5:    NoSimilarShapelets.add(selectShapelet); {添加到不相似的 shapelets 中}
6:    Distance←selectShapelet.splitThreshold; {获取距离阈值}
7:    for j←i+1 to size do {每一个候选 shapelet}
8:      ToPrune←candidateShapelets.get(j); {获取待剪枝的候选 shapelet}
9:      Dist←subdist(selectShapelet,ToPrune); {计算距离}
10:     if (Dist≤Distance) {若两者间的距离小于阈值且类标不同,保留待剪枝的 shapelet}
11:       if (selectShapelet.classLabel!=ToPrune.classLabel)
12:         NoSimilarShapelets.add(ToPrune);
13:     else {若两者间的距离大于阈值,保留待剪枝的 shapelet}
14:       NoSimilarShapelets.add(ToPrune);
15:   candidateShapelets←NoSimilarShapelets;
16:   size←candidateShapelets.size();{重新获取候选 shapelets 的个数}
```

17: **return** *NoSimilarShapelets*;

在循环的开始和结束处,我们都需要初始化候选 shapelets 的数目,因为基本上每一次循环,候选 shapelets 的数目都将会减少(第 2 行和第 16 行).对于候选 shapelets 中的每一个 shapelet,我们都将其与优于它的 shapelets 作比较,并决定是否剪去它(第 3 行~第 14 行).若 *ToPrune* shapelet 和 *selectShapelet* shapelet 间的子序列距离小于等于 *selectShapelet* 的分裂阈值,我们将比较它们的类标:当它们类标相同时,剪去 *ToPrune*;否则,我们保留 *ToPrune*,并将它应用于下一次循环中(第 10 行~第 14 行).最后,我们得到用于 shapelet 覆盖步骤的所有不相似的 shapelets.

2.3 Shapelet覆盖

文献[1]使用 5 折交叉验证方法来确定用于数据转换的 shapelets 的数量.显然,这是一项十分耗时的工作,尤其当训练数据量比较大时,它的缺点会更明显.原因是,此方法需要运行多次才能得到拥有较好分类准确率的 shapelets 的数量.为了有效并简洁地获取用于最终转换的 shapelets 的数量,我们提出了一个新的概念——shapelet 覆盖.

定义 7(Shapelet 覆盖). 给定 shapelet (S_1,τ_1)和与它相对应的 orderline L,分裂阈值 τ_1 将 L 上的数据点或者说数据集 D 分割成两部分:左边是与 shapelet 间子序列距离小于 τ_1 的时间序列数据集 D_l,右边是与 shapelet 间子序列距离大于 τ_1 的时间序列数据集 D_r.那么我们认为,D_l 能够被 shapelet 成功覆盖,D_r 不能够被 shapelet 覆盖.参照图 3,根据此定义,我们可以认为圆形所代表的时间序列是能够被 shapelet 覆盖的数据集.

本文提出的 shapelet 覆盖的概念借鉴了属性选择算法中序列覆盖的思想.这里,我们可以将 shapelet 当做时间序列的一个特殊的特征来处理.两者间的主要区别在:于 shapelet 覆盖是一种近似的覆盖方法,它并不意味着某一时间序列包含此 shapelet,而是它们的子序列距离小于某一给定的阈值.通过 shapelet 覆盖来选择 shapelets 的过程见算法 6.

算法 6. *ShapeletsCoverage*(*NoSimilarShapelets*,δ).

输入:不相似的 Shapelets *NoSimilarShapelets*、覆盖参数δ;

输出:辨别性 shapelets *Shapelets*.

```
1: Shapelets←∅;
2: Initialize(InstanceTable); {初始化 InstanceTable}
3: size←NoSimilarShapelets.size(); {不相似 shapelets 的个数}
4: for i←0 to size do {每一个候选 shapelet}
5:     Selected←false; {标志位}
6:     InstanceID←NoSimilarShapelets.get(i).InstanceID; {获取被覆盖的实例 ID}
7:     for j←0 to InstanceID.size() do {每一个实例 ID}
8:         if (InstanceTable.contains(InstanceID_j)) {InstanceTable 是否包含此 ID}
9:             Selected←true;
10:                if (InstanceID_j.number()<δ) {是否达到覆盖次数δ}
11:                    if (InstanceID_j.number()+1≥δ) {达到覆盖次数δ则移除此 ID,否则加 1}
12:                        InstanceTable.remove(InstanceID_j);
13:                    else
14:                        InstanceID_j.number()++;
15:        if (Selected) {若标志位为真,则加入到辨别性 shapelets 中}
16:            Shapelets.add(NoSimilarShapelets.get(i));
17: return Shapelets;
```

本文的算法通过扫描一次不相似的 shapelets 来得到最终用于转换的辨别性 shapelets(第 4 行).当一个 shapelet 被访问之后,被此 shapelet 覆盖的时间序列将被移除掉.但是在真正的时间序列分类任务中,考虑到分类

准确率,我们可能希望用多个 shapelets 去覆盖一个实例.为实现此想法,引进了一个 shapelet 覆盖参数δ.若一条时间序列被至少δ个 shapelets 覆盖,这条时间序列将不会被进一步考虑(第 8 行~第 14 行).用一个名为 *InstanceTale* 的数组来存储每一条时间序列的计数值.初始时,*InstanceTable* 中的每一个值被初始化为 0(第 2 行).为更加清楚地解释此过程,采用了一个简单的实例进行说明,如图 4 和图 5 所示.

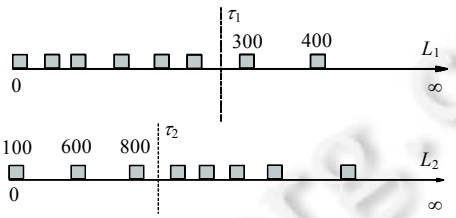

Fig.4 Two orderlines

图 4 两个 orderline 示例

TSID	Count
100	1
200	1
300	0
400	0
500	1
600	1
700	1
800	1

TSID	Count
100	2
200	1
300	0
400	0
500	1
600	2
700	1
800	2

(a) 第 1 次循环时的 *InstanceTable* (b) 第 2 次循环时的 *InstanceTable*

Fig.5 Coverage of time series

图 5 时间序列被覆盖情况示例

在图 4 中,orderline L_1 和 L_2 展示了两个不同 shapelets 区分实例的情况.假设δ=2,第 1 次循环时,第 1 个 shapelet 的 orderline 为 L_1,*InstanceTable* 如图 5(a)所示.时间序列{300,400}与第 1 个 shapelet 之间的距离大于τ,所以它们未被此 shapelet 覆盖,并且它们的计数值仍为 0.由于每个实例的计数值更新之后都没有达到阈值δ=2,所以没有时间序列被移除.在第 2 次循环时,orderline 为 L_2,实例{100,600,800}与该 shapelet 间的距离小于阈值τ,所以它们的计数值都增加 1(如图 5(b)所示).按照前述的规定,时间序列{100,600,800}可以从 *InstanceTable* 中移除.需要注意的是:当且仅当一个 shapelet 能覆盖 InstanceTable 中的任何一个实例时,它才能加入最终的 shapelets 中(第 15 行、第 16 行).

2.4 整合Shapelets剪枝和覆盖方法

在第 2.2 节和第 2.3 节中,首先通过对候选 shapelet 剪枝,获得不相似的 shapelets;然后,根据 shapelet 覆盖方法,从不相似的 shapelets 中得到用于数据转换的辨别性 shapelets.为更快地获取用于数据转换的 shapelets,可考虑将 shapelet 剪枝与 shapelet 覆盖方法相结合,即,直接通过对候选 shapelets 的遍历而得到最终的 shapelets.此部分的目的是将 shapelet 剪枝和 shapelet 覆盖相整合.该方法相比较于前面分别计算 shapelet 剪枝和 shapelet 覆盖的方法至少节省了一个时间复杂度为 $O(m)$的循环,m 为不相似 shapelets 的个数.该整合版本见算法 7.

算法 7. *PruneAndCoverage*(*candidateShapelets*,δ).

输入:候选 shapelets *candidateShapelets*、覆盖参数δ;

输出:辨别性 shapelets *Shapelets*.

1: *Shapelets*←∅;

2: *size*←*candidateShapelets.size*(); {候选 shapelets 的个数}

3: *Initialize*(*InstanceTable*); {初始化 *InstanceTable*}

```
4:  for i←0 to size do {每一个候选 shapelet}
5:    if (InstanceTable.isEmpty()) {若 InstanceTable 为空,跳出循环}
6:      break;
7:    selectShapelet←candidateShapelets.get(i); {获取候选 shapelet}
8:    Shapelets←UpdateCoverage(selectShapelet,δ); {更新 InstanceTable}
9:    for j←i+1 to size do {每一个候选 shapelet}
10:     ToPrune←candidateShapelets.get(j); {获取待剪枝的 shapelet}
11:     PruneShapelets(selectShapelet,ToPrune); {剪枝}
12:   size←UpdateSize();{更新候选 shapelets 的个数}
13: return Shapelets;
```

第 1 行~第 3 行中,首先初始化了 *InstanceTable* 和候选 shapelets 的数量.然后,对于每一个候选的 shapelet,在剪去与它相似的 shapelets 之前,我们需要检查 *InstanceTable* 是否为空:若 *InstanceTable* 为空,由于此方法将不会选择任何 shapelet,循环就可以直接中断(第 5 行、第 6 行).此后,继续更新每一条时间序列的覆盖情况,并剪去相似的 shapelets(第 7 行~第 11 行),具体方法见第 2.2 节和第 2.3 节.当一个候选 shapelet 访问结束后,因为候选 shapelets 的数量可能发生了变化,我们得更新 *size* 的大小(第 12 行).最终返回用于数据转换的 shapelets.

3 实验与评价

本节将评估辨别性 shapelets 在时间序列分类问题上的表现.我们对原有的方法做出了一些改变:

- 首先,将讨论用于数据转换时 shapelet 覆盖参数δ的最优值;
- 其次,将提出的 ShapeletSelection 算法与文献[1]中的 ShapeletFilter 算法、文献[17]中的 ClusterShapelet 算法以及经典的 1-NN 分类器相比较,阐明了辨别性 shapelets 在时间序列分类问题上的优势.所有的算法均是在 Weka 框架下,使用 Java 代码实现的.所采用的数据集均包括训练集和测试集,其中,shapelets 的发现和分类器的构建都是在训练集上进行的,测试集仅用于测试分类器的分类准确率.我们还需要感谢 Lines 提供的关于 ShapeletFilter 的源代码.

3.1 覆盖参数

在时间序列分类任务中,为分类准确率考虑,我们希望用多个 shapelets 去覆盖某个实例.此部分的目标就是找到一个合适的 shapelet 覆盖参数δ.为获取最好的δ,我们首先讨论δ和 shapelets 数量间的关系;然后,试图通过经验估计来设置能够获得最好分类准确率的覆盖参数δ.

图 6 展示了δ变化时,17 个数据集(见表 2)的 shapelets 数目的变化.所有的实验都是在这些数据的训练集上进行的.我们可以很明显地观察到:初始时,shapelets 的数目会随着δ的增长而增长;当 shapelets 的数目无法满足δ的需求时会到达 shapelets 数目的峰值并保持不变.而数据集 *Coffee* 和 *OliveOil* 是两个例外情况,原因是:当覆盖参数δ从 1 增长到 9 时,这两个数据集有足够的非相似 shapelets 来满足覆盖参数的需求.

从图 7 中可以发现:当δ=4 时,大部分数据集的分类准确率接近或达到了他们的最大值.所以,为保持简洁性和一致性,我们在接下来的实验中将覆盖参数δ设置为 4.注意:实验中的所有分类准确率都是通过在训练集上建立分类器,然后在测试集上进行测试所得到的.

另外发现,候选 shapelets 时的两个长度参数 min 和 max 的设置也是一个难题.由于它们定义了候选 shapelets 的长度范围,参数设置不对时可能发现不了最具有辨别性的 shapelet,从而对分类器的准确率造成影响.为包含尽可能长的长度,我们统一将最小长度设置为 $n/11$,最大长度设置为 $n/2$,n 为时间序列中属性的个数.