



JIANGSU NORMAL UNIVERSITY

**课程论文**

**课 程 名 称：**  软件过程管理

**论 文 题 目：** 基于时间序列的计算系统设计与实现

**姓 名：**  盛锦超

**学 院：**  智慧教育学院

**专 业：**  电子信息

**学 号：**  1032012026064

基于时间序列的计算系统设计与实现

盛锦超1

1(江苏师范大学 智慧教育学院,江苏 徐州 221100)

通讯作者: 盛锦超, E-mail: [shengjinchao@qq.com](mailto:shengjinchao@qq.com)

摘 要: 鉴于目前有关时间序列的各种内容中,有关如何计算时间序列的函数及系统没有得到广泛的介绍和应用;时间序列是具有高维度和时间前后相关性的数据点集合,在各种领域都有很重要的作用;本文就时间序列的特点,对时间序列的表示,度量,预测,分类和聚类等内容设计与实现计算时间序列特定要求的函数集合,利用软件工程的思维设计与Python语言进行实现;通过测试证明了计算系统的可行性.尽管系统存在着许多的不足,但最终仍旧得到了有效并可供参考的结果.

关键词: 时间序列;软件工程;Python;机器学习;Shapelet

Design and Implementation of Computing System Based on Time Series

SHENG Jin-Chao1

1(School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221100, China)

**Abstract**: In view of the various contents about time series, the functions and systems about how to calculate time series have not been widely introduced and applied. Time series is a set of data points with high dimension and time correlation, which plays an important role in various fields. In this paper, the characteristics of time series, the representation, measurement and prediction of time series are discussed, In this paper, we design and implement the function set for computing the specific requirements of time series by using the thinking design of software engineering and python language. The feasibility of the computing system is proved by testing. Although the system has many shortcomings, it still gets effective and referential results

**Key words**: time series; software engineering; Python; machine learning; Shapelet

时间序列(Time Series)是按照时间顺序排列而成的一连串的实值数据,其中每个时间点都具有相同的间隔;一般来说,时间序列可以通过传感器或者其他采样工具进行等频率采样;在等额点上,实值数据可能是一维或者多维,甚至在有些情况可能存在随机变量的情况[1];但是大部分情况下是一维且唯一的;时间序列还存在静态和动态的划分,其中静态划分是指时间序列的起始点和终点已经确定,而动态划分则两者都不确定;针对动态划分,一般设置一个滑动窗口在相同频率的情况下进行等额获取.时间序列具有高维度、时间前后相关性和变化幅度大而快的特点且广泛应用于金融、医疗、气象和环境检测等众多领域[2];在知识发现(Knowledge Discovery in Database)、模式识别(Pattern Recognition)和机器学习(Machine Learning)等领域都有着非常不错的发展[3],由于其普遍性及特殊性,受到许多研究者的研究;其研究成果也加深了人们对时间序列的认识与理解.

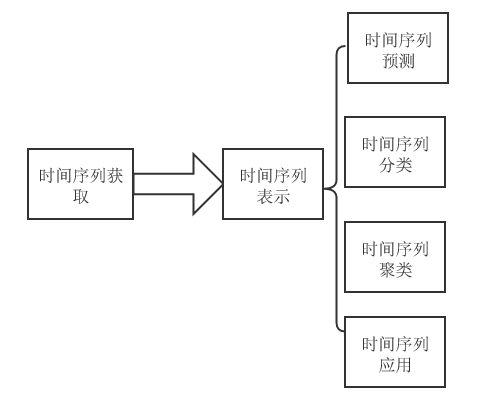


Fig.1 Research direction of time series

图1 时间序列研究方向

时间序列的研究大致如上图1所示.

时间序列的获取使用了传感器技术,获得到的数据一般呈现连续波形特点;对时间序列的表示则是通过对时间序列采用各种办法进行降维,使之能够被表示与应用;一般采用时间序列的表示方法有[4]采样(一般为等额采样),分段聚合近似表示(PAA),分段线性表示方法,基于域方法,符号表示方法(SAX),还有奇异值分解等.

时间序列预测是使用传统的统计学原理进行数值度量分析,比如经典的季节性模型和ARAM模型等;时间序列的分类是对有标签的时间序列进行分类,而时间序列聚类则针对无标签;时间序列的应用是将上述的技术组合起来,帮助人进行决策,主要包括模式识别等.

时间序列的分类算法[4]大致分为基于模型的分类算法,比如贝叶斯模型,马尔可夫模型;但由于模型存在局限性即时间序列的数据值需要满足模型对数据的需要,故此类算法具有局限性;基于全局特征的算法,比如1-NN算法,但其计算量大,不适合巨量数据;基于局部特征的算法,最著名的是Shapelet算法[5],该算法通过寻找时间序列中最具有代表性的子序列表示整条时间序列,有关Shapelet的更多信息会在后面介绍;最后是基于组合的分类算法,实际上就是将上述的算法进行组合,类似于机器学习的集成训练.

时间序列的聚类方法主要分为传统方法和非传统方法;其中传统方法主要是基于划分的聚类K-means,基于层次的分裂和聚合算法,谱聚类等;而非传统以u\_Shapelet[6]最具有解释性,其中对u\_Shapelet的介绍稍候;对u\_Shapelet进行研究并加速的研究非常多,如L[7]使用SAX简化表示Shapelet并用随机掩码筛选百分之1的数据集,得到了与原方法基本相同的准确率并提高了两个数量级的速度.

鉴于目前没有对时间序列计算相关的文献,本文就简要提出一些计算函数,主要包括了预测,分类与聚类;目的可以解决时间序列计算问题;又希望计算函数能够成体系的完成任务,故与软件工程的思维结合,建立一个基于时间序列计算的系统;文章的后续安排如下:第一章会介绍一些概念与公式,为后面的函数设计提供基础;第二章之后会围绕软件工程的最基本思路介绍系统,包括需求工程、系统建模、设计实现、实验测试;第二章介绍需求工程，主要围绕功能需求和可行性分析;第三章介绍系统建模,主要介绍图例;第四章设计实现要求;第五章通过实验测试计算的可能性;第六章总结全文.

# 1 相关工作

## **1.1**时间序列及距离计算方法

**定义1**时间序列:设时间序列T是一条从t1到tn的实值序列,其中n为时间序列的长度;本文的实值为一维实数且为静态时间序列数据

**定义2**时间序列数据集:设D={T1,T2,...,Tm}为具有m条时间序列的集合

**定义**3时间序列子序列:设Si,l={ti,...,tl}为一条时间序列T从i到l距离的序列;显然若T的长度为n,则1≤𝑖≤𝑛,1≤𝑙≤𝑛−𝑖+1

**定义**4时间序列的距离度量:设Ti和Tj是两条时间序列,其距离为dist(Ti,Tj)

根据时间序列的特点,时间序列的距离度量可以分为等长与不等长两种情况;等长距离计算最常见就是欧式距离;但为了实现尺度和偏移不变性,一般需要先对时间序列进行z-规范化处理(1.1),其中μ是整条时间序列的均值,σ是整条时间序列的方差;并且欧氏距离仍旧存在较长的时间序列比较短的时间序列大的情况,故需要对欧氏距离进行长度规范化(1.2).

 (1.1)

 (1.2)

不等长情况下,我们仍旧可以使用欧式距离,但是此时是短距离与长距离不同起点的组合的最小值,见(1.3);此外动态时间规整(DTW)距离则会更好计算,DTW通过建立一个距离矩阵m\*n;其中m和n分别是两条时间序列的长度,通过(1.4)进行计算;其中设置g(0,0)=0,g(0,∞)=0,g(∞,0)=0.

 (1.3)

 (1.4)

## **1.2**计算Shapelet及加速算法

**定义5**Shapelet/u\_Shapelet：Shapelet是一个元组<Si,l,dt>;前者表明其是一个子序列,后者dt是指将子序列与D中的时间序列进行距离计算并得到不同的距离dis,dt将时间序列根据距离dis分为DL和DR;其中DL中的dis≤dt,而DR则与DL相反即dis>dt;显然dt的选择可以有无数值,我们将dis集合排序得到Orderline我们规定dt只能取每相邻两个dis的均值;故共有D-1个dt可选项.

**定义6**衡量Shapelet/u\_Shapelet好坏的指标:gap(1.5)

 (1.5)

根据定义μR和μL分别是R和L的均值,σ同理;当然评价指标还有均方根标准差(RMSSTD)、R²、I等[8];本文不做具体介绍.

因为每个子序列都要与其他的子序列进行比较,所以计算中存在大量的重复;为了降低算法的时间复杂度,Mueen[9]等人提出通过存储一些统计量和重复使用中间的计算结果,把计算规范化欧式距离的时间复杂度由线性级别降低到常数级别.

这里我们只介绍最重要的公式,其他内容可参阅[9].

**定义7**加速计算Cs(1.6):

 (1.6)

需要说明的是X是长度为m的子序列,而Y是长度为n的时间序列,m<n.

## **1.3**时间序列方法简述

本文整合时间序列的三大部分,预测,分类与聚类;其中预测主要见[10];分类见[11];聚类见[12].

由于本文的目的并不是来介绍时间序列的各种算法的,但是本文会对一些算法做出自己的设计;但由于时间有限,故算法的具体细节请参照[2] [5] [6] [7] [9].

对于算法的评价指标,会在后面章节简单介绍,这不是文章的重点.

# 2 需求工程

本文采用传统的软件工程建模方法即瀑布模型;由于工作量的大小,本文会简化模型并按照需求工程、系统建模、设计实现、系统测试四部分组成;具体软件工程思想见[13].

需求工程的目的是想让软件或者业务产生什么效果,本文中需求工程围绕功能需求和可行性分析两部分展开;其中功能需求介绍本文各个计算函数的功能,而可行性分析则通过经济可行性,技术可行性,社会可行性三部分叙述系统的可行性.

## **2.1**功能需求

对于时间序列计算函数,对如何获取时间序列不是本文考虑的范围;相反应该考虑时间序列的表示,度量,分类与聚类,预测这五个主要模块;具体如下图2所示:

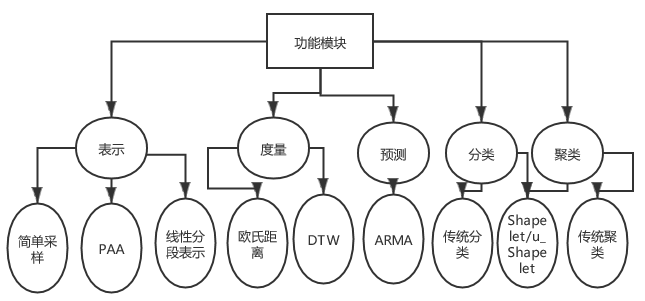


Fig.2 Brief introduction of function modules

图2 功能模块简述

(1)表示模块主要是对时间序列进行降维,分别[4]是简单采样(通过间隔选取时间序列),PAA(通过间隔取平均值进行采样),线性分段表示(通过观察时间序列单个观测点前后两个近邻观测点的大小进行筛选最具有特点的观测点).

(2)度量模块主要根据上述表达为欧式距离和DTW两个距离计算方法;其中DTW更适用于后续功能中的传统方法,而欧氏距离更加适用于Shapelet的方法.

(3)预测模块主要设计ARMA模型;分类和聚类模块主要介绍传统分类的模型与基于Shapelet的方法;其中传统模型可以使用决策树、层次与划分聚类等方法.

当然按照传统的需求工程,还需要介绍非功能性需求;但基于本文设计的思想及内容较为常见且可执行性比较强;且由一个完成,传统的需求工程介绍的各种详细的非功能性需求并不适用于本系统.故不作具体阐述,更多细节可以参见[13].

## **2.2**可行性分析

可行性分析是根据系统的内容,对其是否能够设计与实现的可行性的研究;本文从经济可行性,技术可行性,社会可行性三部分叙述,具体如下:

* 经济可行性:系统进行的大量工作都是在自己的笔记本上进行的,不管是电费还是网费;都是满足开发的成本的;而如果后期系统上线或其他原因需要产生费用,整体来看,不会超出现有经济基本成本即3000RMB以上;故显然满足经济可行性.
* 技术可行性:本文将使用Python语言开发,其基本流程采用传统的MVC模式,稳定性较好;此外Python代码简洁易懂,易用性高;性能和空间足够,存储数据的数据库是SQL SERVER,基本满足信息安全的要求;综上满足技术可行性的需求.
* 社会可行性:系统所使用的大量代码都是Python提供的开源代码和作者自身的手写创作,符合法律和道德需求,对社会不会产生不良影响;故满足社会可行性.

# 3 系统建模

从系统建模的角度,我们更深入介绍各个功能的具体细节;系统开发的基本思路是MVC模式,其中包括了模型M、视图V和控制器C;模型是指业务模型,在本文中主要指各种时间序列的计算函数;视图时指用户界面,主要为了让用户看到,本文主要通过作图给用户展示算法最后的特性;控制器主要保证模型与视图功能同步,这不是本文的介绍重点.本节首先详细介绍本文的模块的建立思路,然后会有一些图的辅助,最后介绍系统的整个循环体系和一些细节与不足遗漏之处.

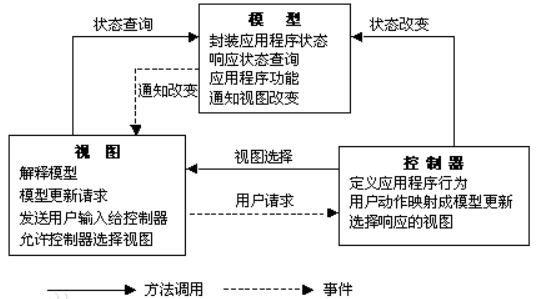


Fig.3 MVC model

图3 MVC模型

## **3.1**模块建立思路

(1)表示模块主要是通过函数对时间序列进行降维计算,降维不但能够降低之后计算的运算时间,更能够提高时间序列的某些显著特点,本文介绍三种基本模块;如上所述,包括了简单采样,PAA和线性分段;其中简单采样和PAA直接返回序列的值,而线性分段则会返回一个index和一个data的元组,故后续的聚类和分类等方法会优先考虑线性分段.

(2)度量模块使用了欧式距离和DTW距离计算,其实度量还有许多关于DTW的变式,如编辑距离;本文仅讨论这两个距离度量原因在于本文提出的分类和聚类算法主要有Shapelet这部分,Shapelet是时间序列的最显著子序列,使用欧氏距离更能体现其局部性;而研究证明DTW对于测量传统聚类算法的距离度量更有优势.

(3)预测模型我们选择的是ARMA模型,但ARMA模型是不需要差分变换的;故最终实际上我们会选择ARIMA模型进行差分变换,它是ARMA模型的变形;但这两种模型都需要对平稳数据进行建模,而时间序列往往都不具备平稳性,故进行一阶差分是有必要的,如果对一阶差分的效果不满意,甚至可以进行二阶差分,多阶差分;ARIMA的公式与细节部分不做阐述,感兴趣可以在中[10]查阅.

预测模型是比较成熟的技术,故使用Python的集成statsmodels.tsa.arima\_model库,需要参数是p,差分次数,q等取1-3左右较为适合的范围;而其相关优化如自相关函数ACF,偏自相关函数PACF等来获得比较好的效果的调整,读者可以查阅相关资料;本文只提供最基础的模型.

(4)分类和聚类模块模型以传统和Shapelet两部分组成,传统模型使用决策树和K-means、层次聚类等组成,通过使用Sk-Learn库进行设计,而Shapelet则通过自己的理解进行计算和设计.

对于分类来说,最主要的考虑如何计算信息增益和计算gap;而聚类则考虑两种情况即是否第一遍全部探索时间序列还是只探索一条时间序列,很显然全探索会增大运算时间同时提高准确率;而单一探索可能会选择了一条信息率不大的时间序列,会对整体时间序列产生不好的结果.

## **3.2**模型图例和系统整体循环

下图4是用例图,主要通过函数处理结果交给需要的用户.

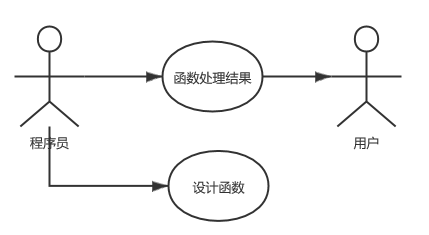


Fig.4 Use case diagram

图4 用例图

下图5是对整个系统进行的分析.

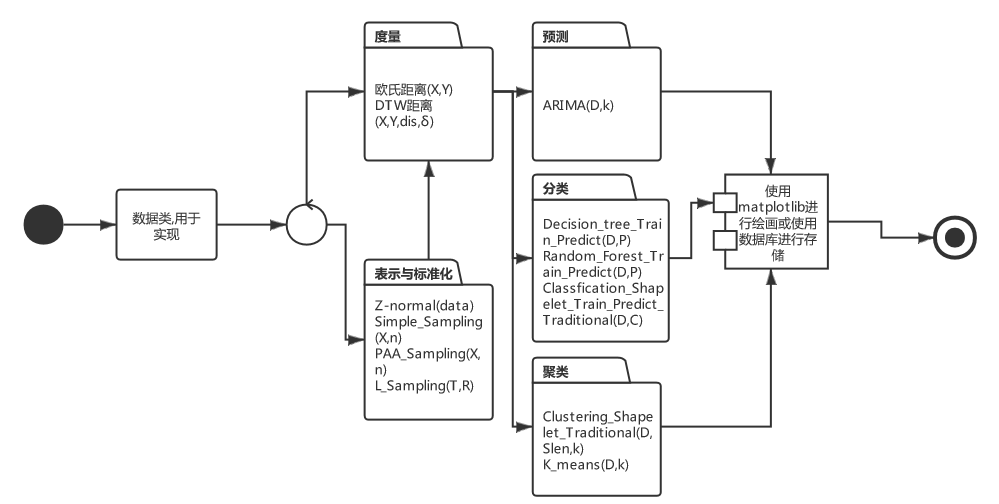


Fig.5 System flow chart

图5 系统流程图

我们详细介绍图5的内容,我们主要设计一个数据类,用于实现各种函数操作,一般来说数据获取到了之后会进行度量;度量模块上会使用表示和标准化模块;预测,分类和聚类模块是在度量模块基础上进行的,最后通过得到的结果进行图示;其中每个模块都有其最核心的函数及参数,具体代码会在下一章给出.

这里需要指出的事情是,虽然本文依照软件工程的思路进行设计;但由于时间和精力有限,本文不能按照既定流程进行更多图例的介绍,此外本文会选择性忽略某些功能,比如数据库部分不在本文的讨论内容里,因此本文并不会进行E-R图的绘制,但是自然的想到得到的结果进行存储也是很重要的事情,简单说明就是可以设置一些属性存储数据和标签,甚至存储模型;这为后续的完善活动提供方向.

最后说明的是,软件工程的UML图例是非常多的,本文涉及的内容非常少,具体可以参阅[13].

# 4 设计实现

本章介绍如何用代码实现算法,再次强调文中大量的数学知识不是本文介绍的重点,故不做具体介绍.

首先本章会列出使用的工具和一些规定,然后对列出关键函数的代码;而下一章会进行测试

**Table 1** System key attributes

表**1** 系统关键属性

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | |
| Python3.6 | | 系统的开发环境 |
| Numpy,Pandas,Sk-learn.etc | | 系统使用的重要库函数 |
| D | | 数据集 |
| P | | 分类标签 |
| X,Y | | 单条数据 |

(1)表示与标准化:

简单采样:

def Simple\_Sampling(X,n):

X = np.array(X)

if len(X)<=n:

return -1

list=[]

for i in range(len(X)):

if i%n == 0:

list.append(X[i])

return list

PAA计算:

def PAA\_Sampling(X,n):

X = np.array(X)

if len(X)<=n:

return -1

list=[]

sum=0

for i in range(len(X)):

sum+=X[i]

if i%n == 0 and i != 0:

sum=sum/n

list.append(sum)

sum=0

return list

线性分段计算:

def L\_Sampling(T,R):

T = np.array(T)

X = []

for i in range(0, len(T)):

X.append((i, T[i]))

vital\_point = []

vital\_point.insert(0, X[0])

index = 0

for i in range(1, len(T) - 1):

if T[i] > T[i - 1] and T[i] > T[i + 1]:

if T[i] / T[index] > R:

index += 1

vital\_point.insert(index, X[i])

if T[i] < T[i - 1] and T[i] < T[i + 1]:

if T[i] == 0 or T[index] / T[i] > R:

index += 1

vital\_point.insert(index, X[i])

index += 1

vital\_point.insert(index, X[len(T) - 1])

return vital\_point

Z-标准化:

def Z\_normal(data):

lenth = len(data)

total = sum(data)

ave = float(total)/lenth

tempsum = sum([pow(data[i] - ave,2) for i in range(lenth)])

tempsum = pow(float(tempsum)/lenth,0.5)

if tempsum==0:

return data

for i in range(lenth):

data[i] = (data[i] - ave)/tempsum

return data

(2)度量

欧式距离:

def Calcs\_Euclidean\_distance(X,Y):

X=np.array(X)

Y=np.array(Y)

if len(X)<=0| len(X)<=0:

return -1

if len(X) == len(Y):

X = Z\_normal(X)

Y = Z\_normal(Y)

dist = np.cumsum([pow(X[i] - Y[i], 2) for i in range(len(X))])

distance = pow(float(dist[len(X)-1]) / len(X), 0.5)

return distance

elif len(X)>=len(Y):

Difference = len(X)-len(Y)

dist=99999

for i in range(Difference):

local=Calcs\_Euclidean\_distance(X[i:len(Y)],Y)

if local<=dist and local>0:

dist=local

return dist

else:

Difference = len(Y) - len(X)

dist = 99999

for i in range(Difference):

local = Calcs\_Euclidean\_distance(Y[i:len(X)+i], X)

if local <= dist and local>0:

dist = local

return dist

DTW计算:

def Calcs\_DTW\_distance(ts\_a, ts\_b, d=lambda x, y: abs(x - y), mww=10000):

ts\_a, ts\_b = np.array(ts\_a), np.array(ts\_b)

M, N = len(ts\_a), len(ts\_b)

cost = np.ones((M, N))

cost[0, 0] = d(ts\_a[0], ts\_b[0])

for i in range(1, M):

cost[i, 0] = cost[i - 1, 0] + d(ts\_a[i], ts\_b[0])

for j in range(1, N):

cost[0, j] = cost[0, j - 1] + d(ts\_a[0], ts\_b[j])

for i in range(1, M):

for j in range(max(1, i - mww), min(N, i + mww)):

choices = cost[i - 1, j - 1], cost[i, j - 1], cost[i - 1, j]

cost[i, j] = min(choices) + d(ts\_a[i], ts\_b[j])

return cost[-1, -1]

(3)预测

ARIMA模型:

def ARIMA\_Forecast(T,k):

model = ARIMA(T, (1, 1, 1)).fit()

result=model.forecast(k)[0]

return result

(4)分类

Shapelet:

def Classfication\_Shapelet\_Train\_Predict\_Traditional(D,C):

D=np.array(D)

P=np.array(D[:,-1])

D=np.array(D[:,0:-1])

ED=Calcs\_ED(P)

for i in range(len(D)):

if C>len(D[i]):

return -1

maxGain=0

maxGap=0

best\_s=[]

dt=0

i1,k1,j1=-1,-1,-1

for i in range(len(D)):

for j in range(C):

for k in range(len(D[i])-j):

list = []

for l in range(len(D)):

sdist=Calcs\_Euclidean\_distance(D[i][k:k+j+1],D[l])

list.append(sdist)

best\_dt,update,Gap,Gain=Calcs\_max\_gain(list,D,P,maxGain,maxGap,ED)

if update:

maxGap=Gap

maxGain=Gain

dt=best\_dt

best\_s=D[i][k:k+j+1]

i1,k1,j1=i+1,k+1,j+1

return best\_s,i1,k1,j1

随机森林:

def Random\_Forest\_Train\_Predict(D,P):

D = pd.DataFrame(D)

P = pd.Series(P)

Target = D.iloc[:, -1]

Data = D.iloc[:, 0:-1]

rfc = RandomForestClassifier(n\_estimators=25, oob\_score=True)

rfc = rfc.fit(Data, Target)

result=rfc.predict(P.values.reshape(len(P),-1))

return result

(5)聚类

K-means:

def K\_means(D,k):

cluster = KMeans(n\_clusters=k, random\_state=0).fit(D)

return cluster.labels\_

U\_Shapelet:

def Clustering\_Shapelet\_Traditional(D,Slen,k):

for i in range(len(D)):

if Slen>len(D[i]):

return -1

U\_shapelets=[]

D=pd.DataFrame(D)

ts=D.iloc[0,:]

while True:

Gap\_and\_Dt=[]

cnt=0

for i in range(Slen):

for j in range(len(ts)-i):

GAP,DT,TS=Calcs\_Gap(D,ts[j:j+i+1],k)

Gap\_and\_Dt.append([GAP,DT,TS])

max=0

index=0

DDt=0

for i in range(len(Gap\_and\_Dt)):

if Gap\_and\_Dt[i][0]>=max:

max=Gap\_and\_Dt[i][0]

index=i

DDt=Gap\_and\_Dt[i][1]

dis = []

for i in range(len(D)):

distance = Calcs\_Euclidean\_distance(Gap\_and\_Dt[index][2], D.iloc[i, :])

dis.append([i, distance])

Da=[]

Danumber=[]

for i in range(len(dis)):

if dis[i][1]<DDt:

Da.append(dis[i])

Danumber.append(dis[i][1])

if len(Da) <=1:

break

maxdis=0

maxindex=0

for i in range(len(dis)):

if dis[i][1]>maxdis:

maxdis=dis[i][1]

maxindex=dis[i][0]

ts =D.iloc[maxindex,:]

cita=np.mean(Danumber)+np.var(Danumber)

P=[]

for i in range(len(dis)):

if dis[i][1]>=cita:

P.append(D.iloc[i,:])

D=pd.DataFrame(P)

这里我们只列出了一些关键函数的核心代码,更多具体细节可参见[14].

# 5 实验测试

本章将对第四章的主要函数进行测试;在机器学习中,分类和聚类是有指标来衡量好坏的;其中分类主要是精确率和召回率,ROC曲线等等;而聚类则是分为外部指标和内部指标,主要有RI、轮廓系数等;虽然本文并不具体介绍这些指标,但这些指标也是本文后续工作开展的方向.

测试将采用黑盒测试,采用的测试环境如下表所示;我们会设立两个数据集,一个是人工数据集,一个是自然数据集;主要用人工数据集进行功能测试,而自然数据集则注重结果测试;在有必要的地方会展示图像.

**Table 2** System environment

表**2** 系统环境

|  |  |  |
| --- | --- | --- |
| 名称 | | 描述 |
| CPU | 英特尔 Core i7-7700HQ | |
| 内存 | 英睿达 DDR4 2400MHz 8GB | |
| 显卡 | Nvidia GeForce GTX 1050 Ti 4GB | |

其中人工数据集在单条选择了600个属性的数据,分为两条data1和data2;而多条数据集则选择基本为20\*6左右的矩阵D1、一个240\*3的矩阵数据集D2和一个7\*20具有显著二分性的数据集D3,具体会微调数据集的大小;自然数据集选取了一组10000\*65的数据集DATA;具体数据集可以在[14]中获取到.

下面对各个模块进行测试,我们测试的具体内容和数据集在测试过程中会进行介绍.测试结果可以保证在其他计算机上有基本相同的结果.

(1)测试表示模块:

由于简单采样和PAA的工作原理是相同的,故对他们首先进行测试,通过人工数据集data1测试这个两个表示函数功能;我们设置的参数间隔为5,可以看到降维后的图形基本保持一致,同时我们可以看到PAA相对于简单采样来说稳定性更好,具体如下图所示:

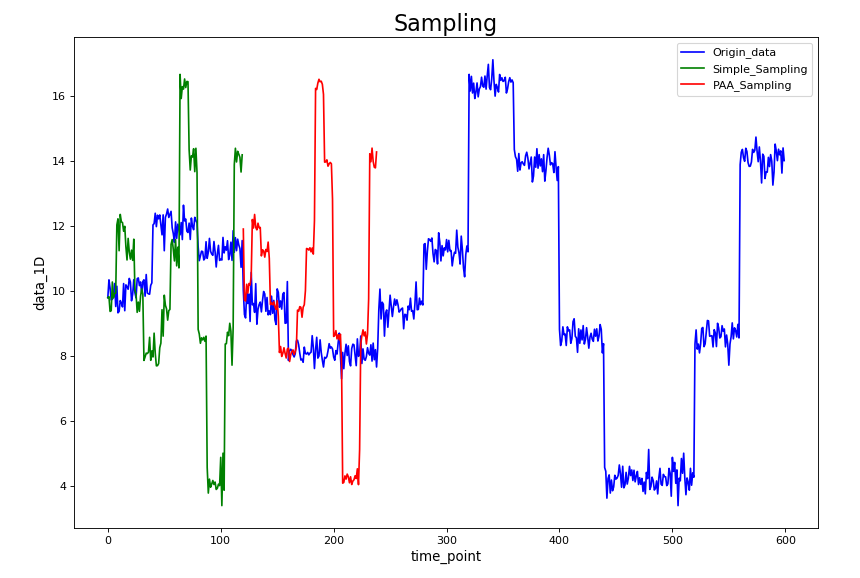


Fig.6 Sampling test chart1

图6 采样测试图1

对于线性分段表示来说,其最主要的特点是会带有点和值的元组组合,通过参数测试与调整,我们发现在L\_Sampling函数中,阈值取到1.05-1.1左右时图像较为明显,如下图所示:

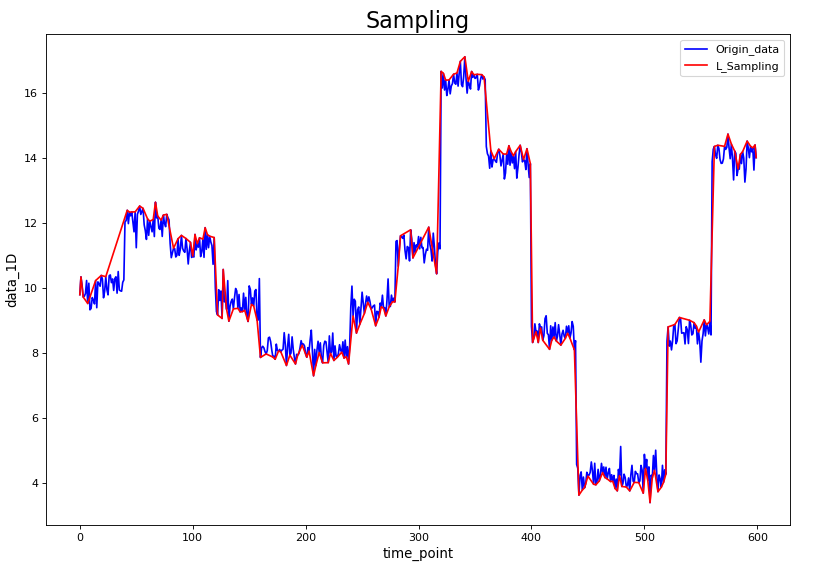


Fig.7 Sampling test chart2

图7 采样测试图2

(2)测试度量模块:

度量模块测试欧氏距离和DTW距离,我们选择data1和data2进行测试;我们测试的思路是首先对等额长度进行计算,然后计算时间;之后对不等长进行计算(长度控制在25%,50%,75%这三个百分比),再计算时间,具体如下表所示:

**Table 3** Distance calculation

表**3** 距离计算

|  |  |  |
| --- | --- | --- |
| 名称 | | 描述 |
| 等长欧式距离 | 距离1.4143时间0.008 | |
| 等长DTW | 距离234.6885时间1.084 | |
| 不等长欧氏距离25% | 距离1.1102\*e-15时间3.907 | |
| 不等长DTW25% | 距离299.3213时间0.217 | |
| 不等长欧氏距离50% | 距离1.005\*e-15时间28.2652 | |
| 不等长DTW25% | 距离219.5045时间0.471 | |
| 不等长欧氏距离75% | 距离1.3605时间22.594 | |
| 不等长DTW75% | 距离280.0526时间0.684 | |

从表中能够看到两种不同的距离度量算法距离的差异性和时间长短;总体上我们看到DTW对处理非等长的数据速度明显大于欧式距离计算,而欧氏距离对等长处理更加擅长.

(3)测试预测模块:

这部分通过ARIMA模型进行测量,ARIMA模型的具体参数与实现细节不做展示(下文同理),通过下图我们能够看到具体的模型成果.其中数据是data1和data2,而红色和绿色是预测结果50个单位长度的走势:

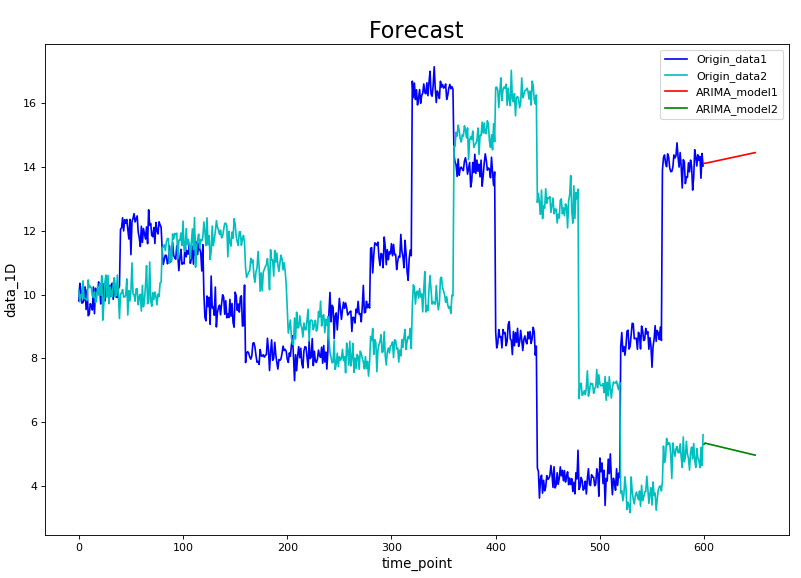


Fig.8 Prediction model test chart

图8 预测模型测试图

(4)测试分类模块:

分类模块主要测试随机森林和Shapelet,我们不加上决策树的原因是随机森林内部就是决策树的实现;我们需要评价指标,故我们选择数据集D1和D2做测试,使用ROC曲线作为随机森林的测试结果,而后者的Shapelet我们测试其是否能够获得最优质的子序列为基准,并计算其运行所消耗的时间,做出分类后的视觉图,观察其分类的效果,当然我们可以通过后续的开发将Shapelet与绘图进行组合.

随机森林的参数只需要数据集D和待分类的没有标签的数据P,获得的标签可以与P的原始标签进行比对,计算相关的指标,具体如下图所示:

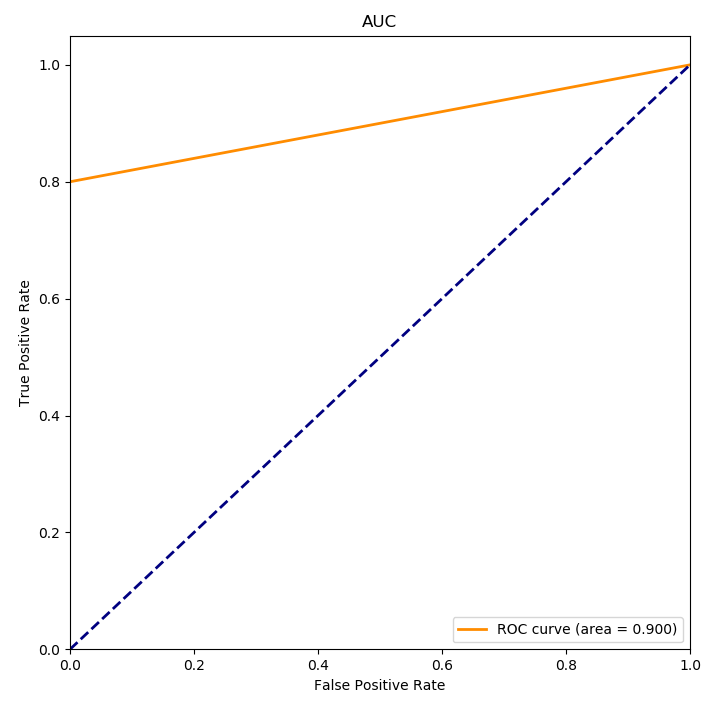


Fig.9 AUC curve of random forest

图9 随机森林AUC曲线

上图选取的是D2数据集,通过AUC曲线可以看到,准确率是比较高的;另外通过设置训练集和验证集后对召回率和精确率可以到达0.8和1.0的数值,这部分内容具体可以见[14].

在D1数据集中使用Classfication\_Shapelet\_Train\_Predict\_Traditional函数,取Slen参数为5,最后获得到了最优Shapelet为第十八条数据从1开始长度为5,其中得到的参数dt为1.377927;而D3数据如下图所示:

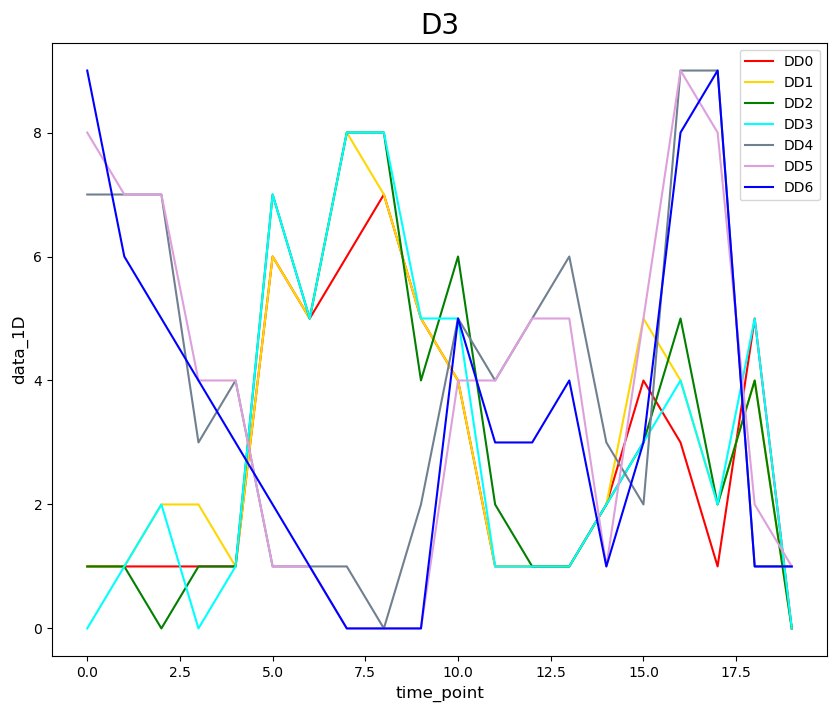


Fig.10 D3 raw data

图10 D3原始数据

D3数据集由上图所示,其中DD0-DD3是属于一类而DD4-DD6属于一类;我们对这个数据集使用Classfication\_Shapelet\_Train\_Predict\_Traditional函数,取Slen参数为10;最终测试时间为25.754,得到的数据为array([1, 1, 1, 1, 6, 5, 6, 7, 5, 4]), 1, 2, 10, 0.6697811566101117;即第一条数据集,从第二个点开始,长度为10,dt为0.6697811566101117;我们对获取到的Shapelet进行分析,通过计算其与数据集的数据,对数据进行分类,如下图所示,大于dt的分为一类,小于dt的分为另一类;最终看到改Shapelet能够将上述的7条数据分成两类,图中红色部分的是Shapelet在数据集中的位置,而黄色和蓝色为两类不同的数据,且判错率在14%;这虽然不是一个很高的水平,但是如果数据量够多,那么经过实验证明[9]判错率会上升,同时注意到的是实验时间是比较久的,主要传统的Shapelet的复杂度就比较高,到达了O(n²m的四次方) [5];但是由于实验环境和时间的问题,我们无法对大量数据进行处理同时加速算法也没有实现,这也可以对以后的完善和加速[9]提供方向.

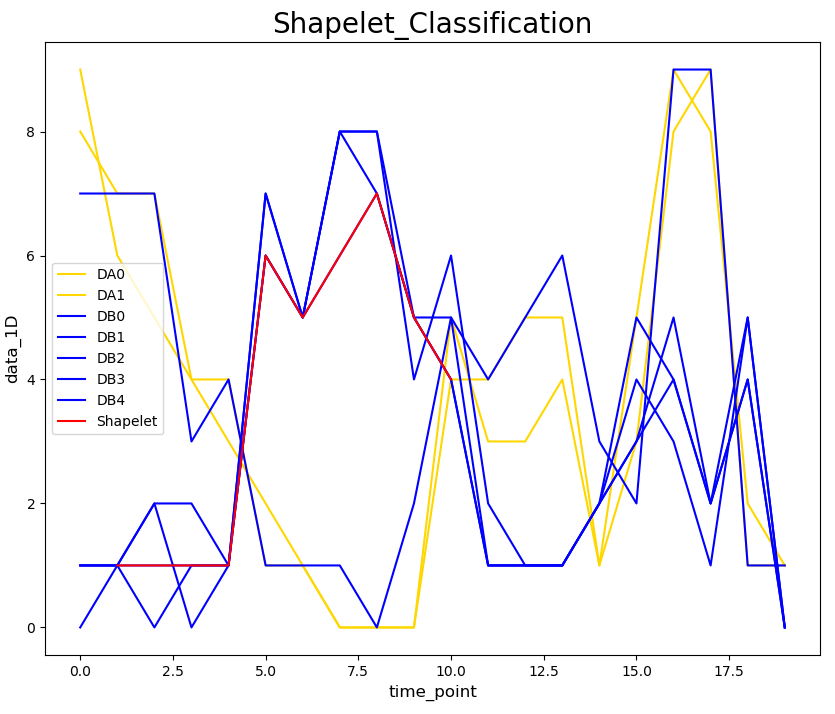


Fig.11 Shapelet classification result chart

图11 Shapelet分类结果图

(5)测试聚类模块:

与分类模块相同,我们主要测试K-means和U\_Shapelet两个内容;聚类的评价指标如上所述,传统的RI指标最为常见,但其实聚类的评价指标特别是内部指标大多因人而异,故参考性没分类那么受到普遍认可.

传统K-means函数我们选择了一个具有787\*2大小的数据,称为D4[14];但我们会首先考虑D2,测试如下图所示:

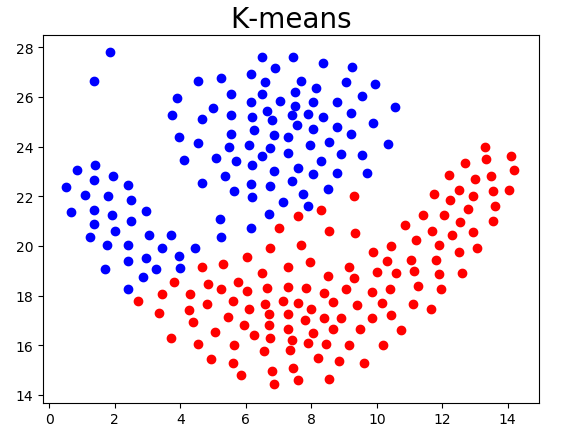


Fig.12 K-means clustering results of D2

图12 D2的K-means聚类结果

我们发现传统的K-means聚类效果并不好,实际商是由于K-means本身的特点造成的,这里如果使用DBSCAN算法[15]可能会得到更好的聚类效果.

这里先介绍评价指标RI,兰德指数,Rand Index;我们首先假设数据已经标注完毕,作为参考簇C,算法聚类形成簇D,样本总数为m.下面以D作为Different的缩写,S为Same的缩写.那么,a=在C中为相同簇在D中为相同簇的样本数量(SS),b=在C中为相同簇在D中为不同簇的样本数量(SD),c=在C中为不同簇在D中为相同簇的样本数量(DS),d=在C中为不同簇在D中为不同簇的样本数量(DD);则RI为下列公式(5-1):

 (5-1)

RI取值范围时[0,1],越接近1聚类效果越好;当然除了RI,聚类的外部评价指标还有ARI,FMI等等,具体可以参见[16].经过计算D2的RI为0.45,D4的RI为0.76,明显好于前者,其图如下所示:

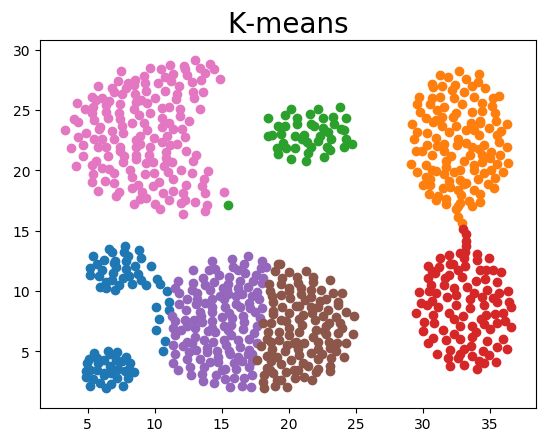


Fig.13 K-means clustering results of D4

图13 D4的K-means聚类结果

U\_Shapelet的测试思路是查找相应的子序列;计算函数的第二个参数是Slen(这里设置了5和10),第三个参数是过滤k值(在这里我们设置为100,实际上根据经验设置5),这些都可以在[6]上查阅.由于删减Da的存在,故U\_Shapelet属于那个原数据并不友好,测试D1(Slen=5)和D3(Slen=10)数据集,分别得到了[1 1 4 5 0]和[1 1 6 5 6 7 5 4 1 1]两条子序列;由于后续操作的内容相同,故只展示D3数据集,如下图所示:

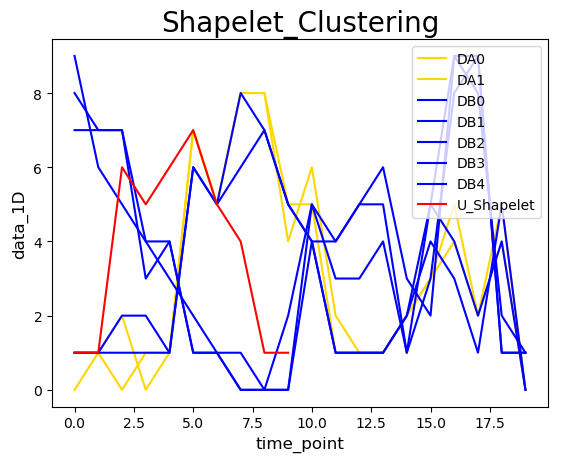


Fig.14 clustering results with U\_Shapelet in D3

图14 D3的U\_Shapelet聚类结果

可以看到相比于分类,聚类测试时间为:2.43415,比分类快,但是聚类的效果不如分类,判错率在28%;与分类同样的道理;聚类效果会在大数据集上变慢,这本质上也是基于其复杂度为O(n²m的四次方),U\_Shapelet[6]提出后,在[7]中得到的加速,但由于其中内容非常多,不仅用到的过滤,还用到的SAX和随机掩码等内容;进一步可以参阅[7].

(6)对大型自然数据集DATA进行测试,得到了如上述测试同样的结果趋势,其中首先对DATA进行降维,采用了PAA方法,参数为100;之后主要对数据进行Shapelet和U\_Shapelet进行了测试,用时分别为1420.759和899.852,分类的判错率为30%,而聚类的RI为0.37,实际上的效果并不能让人满意.

# 6 本文总结

文章的思路一开始通过介绍时间序列的特点,对主流时间序列的一些研究方向和内容进行了概括;然后文章从第一章开始介绍时间序列的表示,之后通过四章,将软件工程的思路带进算法中;对时间序列进行了需求工程、系统建模、设计实现与实验测试;文章围绕时间序列数据集的表示、度量、预测、分类和聚类这五个主题叙述了函数的构建,函数的测试,并通过图像呈现了出来;整体功能基本完成,达到了目的.

文章不足之处在于由于设备和时间等问题,很多内容没有展现出来;比如前端构建没有设计,数据库的设计与存储等;同样由于时间序列的高复杂度特点,我们设计的算法还需要很多的优化来提高速度和准确率,此外算法的具体实现过程及提高的思路过程等信息同样也没有过多涉及.这些都是文章需要修改的地方.

参考文献(References):

1. 左良利. 基于DTW的不确定时间序列分类方法研究[D].南京航空航天大学,2019.
2. 孟庆红. 时间序列聚类中U-shapelets提取方法的研究[D].华东师范大学,2018.
3. 李俊奎. 时间序列相似性问题研究[D].华中科技大学,2008.
4. 嵇存. 基于Shapelet的时间序列分类方法研究[D].山东大学,2017.
5. Ye L , Keogh E J . Time series shapelets: a new primitive for data mining[C]// Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, 2009.
6. Zakaria J, Mueen A, Keogh E. Clustering time series using unsupervised-shapelets[C]//2012 IEEE 12th International Conference on Data Mining. IEEE, 2012: 785-794.
7. Ulanova L, Begum N, Keogh E. Scalable clustering of time series with u-shapelets[C]//Proceedings of the 2015 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2015: 900-908.
8. 余思琴. 基于U-shapelets的时间序列聚类方法研究[D].中国矿业大学,2018.
9. Mueen A , Keogh E , Young N E . Logical-Shapelets: An Expressive Primitive for Time Series Classification[C]// Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, 2011.
10. 张贤达. 时间序列分析:高阶统计量方法[M]. 清华大学出版社, 1996.
11. 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, 000(007):1448-1459.
12. Zhou YM, Xu BW, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Trans. on Software Engineering and Methodology, 2014,23(1):10:110:51.
13. Ian Sommerille. 软件工程(原书第10版)[M]. 机械工业出版社, 2018.
14. https://github.com/a178029910/Software\_engineering\_project/tree/main/Time\_Series\_Code\_Data
15. Kaufman L , Rousseeuw P J . Finding Groups in Data: An Introduction to Cluster Analysis[J]. Wiley, 2005.
16. 高瑞. 方差优化初始聚类中心的K-medoids算法及外部聚类评价指标研究[D]. 2015.