《数据挖掘》实验报告

实验一　数据预处理

专　　业： 计算机科学与技术

班　　级： 计算机 1603 班

学　　号： 1611640305

姓　　名： 温吉祥

完成日期：2019.5.14

1. 问题描述

假定用于分析的数据包含属性 age。数据元组中 age 的值如下（按递增序）：13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70。使用你所熟悉的程序设计语言进行编程，实现如下功能：

(a) 使用按箱平均值平滑法对以上数据进行平滑，箱的深度从键盘输入。

(b) 使用按箱中值平滑法对以上数据进行平滑，箱的深度从键盘输入。

(c) 使用按箱边界值平滑法对以上数据进行平滑，箱的深度从键盘输入。

2. 实验要求

编写程序（编程语言不限），完成如下功能：

(a)读取文本文件中的数据，做分箱处理，箱的深度从键盘输入；输出该组数据按平均值、中值、边界值、中列数平滑的结果。若分箱时末箱数据不足，则用最后之值复制代替。

(b)找出数据中的离群点。

(c)要求程序具有通用性，不能限定数据的个数；从数据文件或数据库中读取数据；程序符合结构化设计风格，关键地方需加注释。

3. 算法分析

1.首先先对数据进行分箱处理，如果数据个数不能整除箱深，则需要对数组进行填充处理。

2.对分箱后的数据进行平滑处理，在这里我使用了 Numpy 二维数组，数组维度表示为（箱数，箱深）。以下是我对四种平滑处理的大致做法：

2.1. 均值平滑：计算每行（每箱）的均值，然后进行逐列替换

2.2. 中位数平滑：计算每行（每箱）的中位数，然后进行逐列替换，并利用平滑前后的差值找出离群点

2.3. 边界值平滑：先得到左右边界的值，然后中间需要平滑的列与左右边界值求绝对差值，并向差值小的边界值平滑，这里还考虑了左右边界差值相等的时候默认按左边界值平滑。

2.4. 中列数平滑：计算每行（每箱）的中列数，然后进行逐列替换

在对特殊数值调用四舍五入 round()方法时需要注意精度的影响，
比如 round(4.5) → 4,round(3.5) → 4。我使用了对数值加 0.5 取整的方法解决。
比如 int(4.5+0.5) →5,int(3.5+0.5) → 4。

4. 算法流程

4.1：导入需要用的包和设置参数

```
import numpy as np
import copy
data = np.loadtxt('./data.txt')
#为防止数据格式问题，统一把数据转化成列表形式
data = list(data)
```

```
print("请输入箱的深度")
depth = int(input())
if depth < 1 or depth > len(data):
        raise ValueError("请输入有效整数")
threshold = 10   #设置离群点阈值
```

4.2：定义一个分箱平滑类，类中实现分箱和几种平滑处理方法

```
#分箱平滑类
class Box_Smoothing:
    def __init__(self,data,depth,threshold):
        self.data = data              #未分箱数据
        self.depth = depth            #箱子深度
        self.threshold = threshold  #离群点阈值
```

4.2.1:分箱

```
    #该函数将原始数据进行分箱
    #该函数将原始数据进行分箱
    def Bins(self):
        length = len(self.data)  #数据的个数
        #算出箱子的个数
        if length%self.depth == 0:
            bins = length//self.depth
        else:
            bins = length//self.depth + 1
        #可能最后一个箱子数据不够，所以需要填充数据
        diff = bins*self.depth - length
        #填充若干个数据
        for i in range(diff):
            data.append(data[-1])
        #构造分箱后的数组,维度为：(箱数，箱深)
        array = np.array(self.data).reshape(bins,self.depth)
        #返回分箱后的结果
        return array
```

4.2.2:按均值平滑

```
    #法一：按均值平滑，并返回离群点
    def Smoothing_By_Mean(self):
        #得到分箱后的数据
        array = self.Bins()
        #计算每行(每个箱子)的均值，并四舍五入
        row_mean = array.mean(axis=1)+0.5
        row_mean = [int(x) for x in row_mean]
```

```
        #进行均值平滑
        for i in range(self.depth):
            array[:,i] = row_mean
        print('----------按均值平滑法----------')
        print(array)
```

4.2.3:按中位数平滑

```
    #法二：按中位数平滑
    def Smoothing_By_Median(self):
        #得到分箱后的数据
        array = self.Bins()
        #深拷贝原数组
        copy_array = copy.deepcopy(array)
        #得到每行(每个箱子)的中位数，并四舍五入
        row_median = np.median(array,axis=1)+0.5
        row_median = [int(x) for x in row_median]
        #进行中位数平滑
        for i in range(self.depth):
            array[:,i] = row_median
        print('----------按中位数平滑法---------')
        print(array)
        #计算平滑后数组与原数组的绝对差值
        diff = abs(array - copy_array)
        #如果差值大于等于 10，则为离群点
        outlier = copy_array[diff>=self.threshold]
        #存在离群点则打印出来
        if outlier.size > 0:
            print('离群点为:' ,outlier)
```

4.2.4:按边界值平滑

```
    #法三：按边界值平滑
    def Smoothing_By_Boundary(self):
        #得到分箱后的数据
        array = self.Bins()
        #左右边界值
        left,right = array[:,0],array[:,-1]
        #需要平滑的列数
        diff = self.depth - 2
        #进行按边界值平滑，平滑差值小的边界值，如果左右边界差值相等，默认选择左
边界值
        for col in range(1,diff+1):
            t = array[:,col]          #取出当前列
            diff_left = abs(t-left)   #当前列与左边界的绝对差值
```

```
            diff_right = abs(t-right) #当前列与右边界的绝对差值
            #标记，如果当前列与左边界的差值 <= 当前列与右边界的差值，则标记为1，否则标记为0
            flag = np.where(diff_left <= diff_right,1,0)
            #对当前列的每一个元素进行边界平滑
            for idx in range(len(t)):
                #如果当前索引下标记为1，则向左边界值平滑，否则向右边界值平滑
                t[idx] = left[idx] if flag[idx] == 1 else right[idx]
        #因为是原地操作，所以直接返回原数组即可
        print('----------按边界值平滑法---------')
        print(array)
```

4.2.5:按中列数平滑

```
    #法四：按中列数平滑
    def Smoothing_By_Midran(self):
        #得到分箱后的数据
        array = self.Bins()
        #计算每行(每个箱子)的中列数，并四舍五入
        row_midran = (array[:,0]+array[:,-1])/2+0.5
        row_midran = [int(x) for x in row_midran]
        #进行均值平滑
        for i in range(self.depth):
            array[:,i] = row_midran
        print('----------按中列数平滑法---------')
        print(array)
```

4.3:调用类输出结果

```
    rst = Box_Smoothing(data,depth,threshold)
    print('--------分箱后未平滑的数据--------')
    print(rst.Bins())
    #按均值平滑，并输出离群点
    rst.Smoothing_By_Mean()
    print('--------------------------------')
    #按中位数平滑
    rst.Smoothing_By_Median()
    print('--------------------------------')
    #按边界值平滑
    rst.Smoothing_By_Boundary()
    print('--------------------------------')
    #按中列数平滑
    rst.Smoothing_By_Midran()
```

5. 编程语言与开发环境

编程语言：Python3.7
开发环境：Windows , Spyder

6. 实验数据

13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70。

7. 运行结果（截图）

这里贴出了箱深为 3 的结果，并找出了离群点 70。其他箱深也能成功运行

```
In [82]: runfile('E:/document/Pyth
数据文件/分箱.py', wdir='E:/documer
代码及数据文件')
请输入箱的深度(1-27)

3
--------分箱后未平滑的数据--------
[[13. 15. 16.]
 [16. 19. 20.]
 [20. 21. 22.]
 [22. 25. 25.]
 [25. 25. 30.]
 [33. 33. 35.]
 [35. 35. 35.]
 [36. 40. 45.]
 [46. 52. 70.]]
----------按均值平滑法----------
[[15. 15. 15.]
 [18. 18. 18.]
 [21. 21. 21.]
 [24. 24. 24.]
 [27. 27. 27.]
 [34. 34. 34.]
 [35. 35. 35.]
 [40. 40. 40.]
 [56. 56. 56.]]
--------------------------------
```

```
----------按中位数平滑法---------
[[15. 15. 15.]
 [19. 19. 19.]
 [21. 21. 21.]
 [25. 25. 25.]
 [25. 25. 25.]
 [33. 33. 33.]
 [35. 35. 35.]
 [40. 40. 40.]
 [52. 52. 52.]]
离群点为: [70.]
---------------------------------
----------按边界值平滑法---------
[[13. 16. 16.]
 [16. 20. 20.]
 [20. 20. 22.]
 [22. 25. 25.]
 [25. 25. 30.]
 [33. 33. 35.]
 [35. 35. 35.]
 [36. 36. 45.]
 [46. 46. 70.]]
---------------------------------
----------按中列数平滑法---------
[[15. 15. 15.]
 [18. 18. 18.]
 [21. 21. 21.]
 [24. 24. 24.]
 [28. 28. 28.]
 [34. 34. 34.]
 [35. 35. 35.]
 [41. 41. 41.]
 [58. 58. 58.]]
```

8. 程序源码及注释

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 14 14:25:26 2019

@author: WinJX
"""

import numpy as np
import copy
class Box_Smoothing:
    def __init__(self,data,depth,threshold):
        self.data = data            #未分箱数据
        self.depth = depth          #箱子深度
        self.threshold = threshold  #离群点阈值

    #该函数将原始数据进行分箱
    def Bins(self):
        length = len(self.data)  #数据的个数
        #算出箱子的个数
        if length%self.depth == 0:
            bins = length//self.depth
        else:
            bins = length//self.depth + 1
        #可能最后一个箱子数据不够，所以需要填充数据
        diff = bins*self.depth - length
        #填充若干个数据
        for i in range(diff):
            data.append(data[-1])

        #构造分箱后的数组,维度为：(箱数，箱深)
        array = np.array(self.data).reshape(bins,self.depth)
        #返回分箱后的结果
        return array

    #法一：按均值平滑，并返回离群点
    def Smoothing_By_Mean(self):
        #得到分箱后的数据
        array = self.Bins()
        #计算每行(每个箱子)的均值，并四舍五入
        row_mean = array.mean(axis=1)+0.5
        row_mean = [int(x) for x in row_mean]
```

```
        #进行均值平滑
        for i in range(self.depth):
            array[:,i] = row_mean

        print('----------按均值平滑法----------')
        print(array)

    #法二：按中位数平滑
    def Smoothing_By_Median(self):
        #得到分箱后的数据
        array = self.Bins()
        #深拷贝原数组
        copy_array = copy.deepcopy(array)
        #得到每行(每个箱子)的中位数，并四舍五入
        row_median = np.median(array,axis=1)+0.5
        row_median = [int(x) for x in row_median]
        #进行中位数平滑
        for i in range(self.depth):
            array[:,i] = row_median
        print('----------按中位数平滑法---------')
        print(array)
        #计算平滑后数组与原数组的绝对差值
        diff = abs(array - copy_array)
        #如果差值大于等于 10，则为离群点
        outlier = copy_array[diff>=self.threshold]
        #存在离群点则打印出来
        if outlier.size > 0:
            print('离群点为:' ,outlier)


    #法三：按边界值平滑
    def Smoothing_By_Boundary(self):
        #得到分箱后的数据
        array = self.Bins()
        #左右边界值
        left,right = array[:,0],array[:,-1]
        #需要平滑的列数
        diff = self.depth - 2
        #进行按边界值平滑，平滑差值小的边界值，如果左右边界差值相等，默认选择左
边界值
        for col in range(1,diff+1):
            t = array[:,col]          #取出当前列
            diff_left = abs(t-left)   #当前列与左边界的绝对差值
            diff_right = abs(t-right) #当前列与右边界的绝对差值
```

```
            #标记，如果当前列与左边界的差值 <= 当前列与右边界的差值，则标记为1，否则标记为0
            flag = np.where(diff_left <= diff_right,1,0)
            #对当前列的每一个元素进行边界平滑
            for idx in range(len(t)):
                #如果当前索引下标记为1，则向左边界值平滑，否则向右边界值平滑
                t[idx] = left[idx] if flag[idx] == 1 else right[idx]
        #因为是原地操作，所以直接返回原数组即可
        print('----------按边界值平滑法---------')
        print(array)

    #法四：按中列数平滑
    def Smoothing_By_Midran(self):
        #得到分箱后的数据
        array = self.Bins()
        #计算每行(每个箱子)的中列数，并四舍五入
        row_midran = (array[:,0]+array[:,-1])/2+0.5
        row_midran = [int(x) for x in row_midran]
        #进行均值平滑
        for i in range(self.depth):
            array[:,i] = row_midran

        print('----------按中列数平滑法---------')
        print(array)

if __name__ == '__main__':
#data = [13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25,
#        30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70]

    data = np.loadtxt('./data.txt')
    #为防止数据格式问题，统一把数据转化成列表形式
    data = list(data)
    print("请输入箱的深度({0}-{1})".format(1,len(data)))
    depth = int(input())
    if depth < 1 or depth > len(data):
        raise ValueError("请输入有效整数")
    threshold = 10   #设置离群点阈值
    rst = Box_Smoothing(data,depth,threshold)
    print('--------分箱后未平滑的数据--------')
    print(rst.Bins())
    #按均值平滑，并输出离群点
    rst.Smoothing_By_Mean()
    print('--------------------------------')
    #按中位数平滑
```

```
rst.Smoothing_By_Median()
print('---------------------------------')
#按边界值平滑
rst.Smoothing_By_Boundary()
print('---------------------------------')
#按中列数平滑
rst.Smoothing_By_Midran()
```

9. 心得体会

NumPy 在数据处理方面确实很强大，一开始还考虑过用 pandas 实现，但因为数据信息比较简单还是用 NumPy 更直接方便。在分箱的时候需要考虑数据能否等分的问题，如果不能还需要进填充。在找离群点的时候巧妙地应用了掩码作为数组索引，包括还利用了 NumPy 数组的广播机制求差值等等。利用面向对象的形式将不同的方法封装能提高代码的可读性和便于修改。