特征处理

问题描述

**我们提供了在双11促销期间一组商人他们相应增长的买家。你的任务是预测未来这些新买家是否会成为这些商家的忠诚客户。换句话说，您需要预测这些新购买者在6个月内再次从同一商家购买商品的可能性。**

特征处理包括数据预处理和特征工程。

数据处理含了数据清洗和数据采样。

数据清洗：1.检测异常样本2.缺省值处理：①缺省值极多②非连续特征缺省值适中③连续特征缺省值适中④缺省较少

数据采样：1. 从负样本中抽取部分样本出来和正样本结合

（欠采样，容易造成信息损失）

2 . 正样本重复若干次

（上采样，保留的数据信息但是有可能放大其噪声数据）。

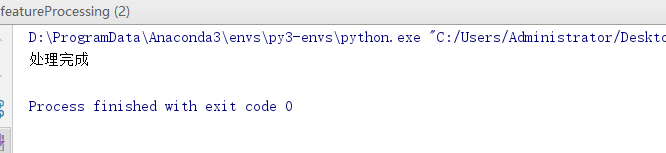
3. 代价敏感学习Cost Sensitive Learning

4. SMOTE即合成少数类过采样技术

代码:

**import** numpy **as** np  
**import** xlrd  
**import** xlwt  
*# from sklearn import preprocessing # 进行标准化数据时，需要引入这个包***from** sklearn.model\_selection **import** train\_test\_split  
  
**def** open\_excel(file):  
 **try**:  
 data = xlrd.open\_workbook(file)  
 **return** data  
 **except** Exception **as** e:  
 print(str(e))  
  
**def** split\_age\_range(age):  
 *"""  
 将特征值年龄进行离散化为8个特征值* **:param** *age: 年龄区间值* **:return***: 离散化后的特征  
 """* **if** age == 0:  
 **return** [1,0,0,0,0,0,0,0,0]  
 **elif** age == 1:  
 **return** [0,1,0,0,0,0,0,0,0]  
 **elif** age == 2:  
 **return** [0,0,1,0,0,0,0,0,0]  
 **elif** age == 3:  
 **return** [0,0,0,1,0,0,0,0,0]  
 **elif** age == 4:  
 **return** [0,0,0,0,1,0,0,0,0]  
 **elif** age == 5:  
 **return** [0,0,0,0,0,1,0,0,0]  
 **elif** age == 6:  
 **return** [0,0,0,0,0,0,1,0,0]  
 **elif** age == 7 **or** age == 8:  
 **return** [0,0,0,0,0,0,0,1,0]  
 *# elif age == 8:  
 # return [0,0,0,0,0,0,0,0,1]***def** split\_gender(gender):  
 *"""  
 将特征值性别进行离散化* **:param** *gender:* **:return***: 返回离散化的特征  
 """* **if** gender == 0:  
 **return** [1,0,0]  
 **elif** gender == 1:  
 **return** [0,1,0]  
*# elif gender == 2:  
# return [0,0,1]***def** split\_log(Log):  
 *"""  
 分割数据文件中的Log数据* **:param** *Log: Log数据* **:return***: 处理后的特征值  
 """* items = Log.strip().split(**'#'**)  
 purchase = 0;total = 0  
 click = 0;add\_to\_card = 0;add\_to\_favourite = 0  
 **for** i **in** range(len(items)):  
 total += 1  
 item = items[i].strip().split(**':'**)  
 **if** item[4] == **'2'**:  
 purchase += 1  
 **if** item[4] == **'1'**:  
 add\_to\_card += 1  
 **if** item[4] == **'3'**:  
 add\_to\_favourite += 1  
 **return** [float(total),float(round(purchase/total,3)),float(add\_to\_card),float(add\_to\_favourite)]  
  
**def** loadDataSet(path, training\_sample,colnameindex=0,by\_name=**u'Sheet1'**):  
 *"""  
 加载数据* **:param** *path: 数据文件存放路径* **:param** *training\_sample: 数据文件名* **:param** *colnameindex: 文件列名下标* **:param** *by\_name: 表名* **:return***: 数据集和类别标签  
 """* dataMat = [];  
 labelMat = [] *# 定义列表* filename = path + training\_sample  
 data = open\_excel(filename)  
 table = data.sheet\_by\_name(by\_name) *# 获得表格* nrows = table.nrows *# 拿到总共行数* colnames = table.row\_values(colnameindex) *# 某一行数据 ['user\_id', 'age\_range', 'gender', 'merchant\_id','label']* **for** rownum **in** range(1, nrows): *# 也就是从Excel第二行开始，第一行表头不算* row = table.row\_values(rownum)  
 **if** row[1] == **'' or** row[2] == **'' or** row[5] == **''**:  
 **continue  
 if** row:  
 app = []  
 app = split\_age\_range(row[1])+split\_gender(row[2]) + split\_log(row[5]) *# 将Log转化为特征值* dataMat.append(app)  
 labelMat.append(float(row[4])) *# 获取类别标签* **return** dataMat, labelMat  
  
**def** main():  
 *"""  
 主函数* **:return***: null  
 """* wb = xlwt.Workbook()  
 ws = wb.add\_sheet(**'Sheet1'**,cell\_overwrite\_ok=**True**)  
 path = **"D:\\AI\\"** training\_sample = **'新建 Microsoft Excel 工作表.xlsx'** *# 训练数据文件* trainingSet, trainingLabels = loadDataSet(path, training\_sample) *# 取训练数据  
 # print(len(trainingSet))* num = len(trainingSet)  
 **for** i **in** range(num):  
 **for** j **in** range(16):  
 ws.write(i,j,trainingSet[i][j])  
 ws.write(i,j+1,trainingLabels[i])  
 wb.save(**'D:\\featuredata.xlsx'**)  
 print(**"处理完成"**)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **"""  
 程序入口  
 """** main()

**运行结果：**



1.该实验主要用到xlrd和xlwt两个库

其中xlrd是读excel，xlwt是写excel的库

2. data = xlrd.open\_workbook(filename)#文件名以及路径，如果路径或者文件名有中文给前面加一个r拜师原生字符。  
table = data.sheets()[0]          #通过索引顺序获取  
  
table = data.sheet\_by\_index(sheet\_indx)) #通过索引顺序获取  
  
table = data.sheet\_by\_name(sheet\_name)#通过名称获取  
  
以上三个函数都会返回一个xlrd.sheet.Sheet()对象

3. 获取整行和整列的值（数组）

table.row\_values(i)

table.col\_values(i)

获取行数和列数

nrows = table.nrows

ncols = table.ncols

循环行列表数据

for i in range(nrows ):

print table.row\_values(i)

单元格

cell\_A1 = table.cell(0,0).value

cell\_C4 = table.cell(2,3).value

使用行列索引

cell\_A1 = table.row(0)[0].value

cell\_A2 = table.col(1)[0].value

简单的写入

row = 0

col = 0

# 类型 0 empty,1 string, 2 number, 3 date, 4 boolean, 5 error

ctype = 1 value = '单元格的值'

xf = 0 # 扩展的格式化

table.put\_cell(row, col, ctype, value, xf)

table.cell(0,0) #单元格的值'

table.cell(0,0).value #单元格的值'

numpy提供了numpy.concatenate((a1,a2,...), axis=0)函数。能够一次完成多个数组的拼接。其中a1,a2,...是数组类型的参数

示例3：

>>> a=np.array([1,2,3])

>>> b=np.array([11,22,33])

>>> c=np.array([44,55,66])

>>> np.concatenate((a,b,c),axis=0) # 默认情况下，axis=0可以不写

array([ 1, 2, 3, 11, 22, 33, 44, 55, 66]) #对于一维数组拼接，axis的值不影响最后的结果

>>> a=np.array([[1,2,3],[4,5,6]])

>>> b=np.array([[11,21,31],[7,8,9]])

>>> np.concatenate((a,b),axis=0)

array([[ 1, 2, 3],

[ 4, 5, 6],

[11, 21, 31],

[ 7, 8, 9]])

>>> np.concatenate((a,b),axis=1) #axis=1表示对应行的数组进行拼接

array([[ 1, 2, 3, 11, 21, 31],

[ 4, 5, 6, 7, 8, 9]])

对numpy.append()和numpy.concatenate()两个函数的运行时间进行比较

示例4：

>>> from time import clock as now

>>> a=np.arange(9999)

>>> b=np.arange(9999)

>>> time1=now()

>>> c=np.append(a,b)

>>> time2=now()

>>> print time2-time1

28.2316728446

>>> a=np.arange(9999)

>>> b=np.arange(9999)

>>> time1=now()

>>> c=np.concatenate((a,b),axis=0)

>>> time2=now()

>>> print time2-time1

20.3934997107

可知，concatenate()效率更高，适合大规模的数据拼接