# 附录

代码清单4-1 基于去跟踪自编码器的填补法

import tensorflow as tf

import numpy as np

class TRAE():

def \_\_init\_\_(self, layer\_size, learning\_rate, momentum, max\_epoch=1000, batch\_size=1):

# 输入

# layer\_size 网络模型各层中的神经元数量

# learning\_rate 学习率

# momentum 动量

# max\_epoch 最大训练次数

# batch\_size 批次

# 对模型的各类超参数进行赋值

self.layer\_size = layer\_size

self.learning\_rate = learning\_rate

self.momentum = momentum

self.max\_epoch = max\_epoch

self.batch\_size = batch\_size

# 调用各类函数构建网络模型

self.define\_variables() # 定义网络输入和期望输出

self.build\_layers() # 定义网络结构

self.build\_loss() # 建立代价函数

self.build\_optimizer() # 创建优化器

# 定义网络输入和期望输出

def define\_variables(self):

self.inputs = tf.placeholder(tf.float32,

shape=[None, self.layer\_size[0]], name='inputs')

self.targets = tf.placeholder(tf.float32,

shape=[None, self.layer\_size[-1]], name='targets')

# 定义网络模型，即规划每层的神经元数量，及设计神经元的计算规则

def build\_layers(self):

s = self.layer\_size[0] # 输入层神经元个数

# 定义网络的权重和阈值参数

def get\_weight\_threshold(n1, n2): # 输入为权重矩阵的大小

w = tf.Variable(tf.truncated\_normal([n1, n2], stddev=0.1,

dtype=tf.float32), name='weight')

b = tf.Variable(tf.zeros([n2]), dtype=tf.float32, name='threshold')

return w, b # 返回权重和阈值

# 建立第一个隐藏层，动态调整神经元输入结构，即依次剔除每个输入属性，并计算神经元输出

def input\_hidden\_layer(n1, n2): # 输入输入层神经元个数，首个隐藏层神经元个数

w, b = get\_weight\_threshold(n1, n2)

sum = tf.matmul(self.inputs, w) + b # 计算输入和阈值的加权和

for j in range(s):

# 对输入和权重进行切片操作

x\_j = tf.slice(self.inputs, [0, j], [-1, 1]) # 分别取每一列属性

w\_j = tf.slice(w, [j, 0], [1, -1]) # 取这一列属性对应的权重

# 从加权和中依次剔除每个输入属性的影响

sum\_j = tf.subtract(sum, tf.matmul(x\_j, w\_j))

sum\_concat = sum\_j if j == 0 else tf.concat([sum\_concat, sum\_j],1)

net = tf.nn.sigmoid(sum\_concat)

return net # 返回当前层的输出

# 定义其他隐藏层，上层隐藏层神经元向当前隐藏层中神经元传递多个输出值

def hidden\_layer(inputs, n1, n2):

# 输入：上层传入的输入，上层隐藏层的神经元个数，当前隐藏层的神经元个数

w, b = get\_weight\_threshold(n1, n2)

for j in range(s):

# 对输入进行切片操作

net\_1j = tf.slice(inputs, [0, j \* n1], [-1, n1])

sum\_j = tf.matmul(net\_1j, w) + b

sum\_concat = sum\_j if j == 0 else tf.concat([sum\_concat, sum\_j],1)

net = tf.nn.sigmoid(sum\_concat)

return net, sum\_concat # 返回神经元输出，激活函数作用前的计算值

# 定义输出层，最后一层隐藏层向输出层神经元正确交付剔除相应输入后的计算结果

def hidden\_output\_layer(inputs, n1, n2):

\_, sum\_concat = hidden\_layer(inputs, n1, n2)

for j in range(s):

y\_j = tf.slice(sum\_concat, [0, j \* n2 + j], [-1, 1])

y = y\_j if j == 0 else tf.concat([y, y\_j], 1)

return y # 返回网络输出

# 依次构建网络模型的各层

net\_j = input\_hidden\_layer(self.layer\_size[0], self.layer\_size[1])

for l in range(1, len(self.layer\_size)-2): # 循环构建隐藏层

net\_j, \_ = hidden\_layer(net\_j, self.layer\_size[l],

self.layer\_size[l + 1])

self.outputs = hidden\_output\_layer(net\_j, self.layer\_size[-2],

self.layer\_size[-1])

# 定义代价函数，利用网络输出和期望输出之间的均方根误差构建代价函数

def build\_loss(self):

self.loss = 0.5 \* tf.reduce\_mean(tf.reduce\_sum(

tf.square(tf.subtract(self.inputs, self.outputs)), 1))

# 定义优化方法，采用结合动量的梯度下降法对代价函数进行优化

def build\_optimizer(self):

self.optimizer = tf.train.MomentumOptimizer(self.learning\_rate,

self.momentum).minimize(self.loss)

代码清单5-1 两阶段式填补法

# 训练阶段，根据训练集训练模型参数

def train(model, train\_x, train\_y, save\_path):

# 输入

# model 网络模型

# train\_x 训练集的输入，二维矩阵

# train\_y 训练集的输出，二维矩阵

# save\_path：练后模型在本地的保存位置，例如 ‘./MODEL/’

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer()) # 初始化网络参数

for step in range(1, model.max\_step+1):

tot\_loss, l = 0, 0 # tot\_loss记录每次训练时的损失值，l为累加变量

# 产生训练批次

generator = batch\_generator\_index(len(train\_x), model.batch\_size)

for indexes in generator:

x, y = train\_x[indexes], train\_y[indexes]

feed = {model.inputs: x, model.targets: y} # 喂入当前批次的训练样本

batch\_loss, \_ = sess.run([model.loss, model.optimizer],

feed\_dict=feed)

tot\_loss, l = tot\_loss + batch\_loss, l + 1

if step == 1 or step % 100 == 0: # 每训练100次，显示当前训练进度

print('step: {}/{}... '.format(step, model.max\_step),

'loss: {:.8f}... '.format(tot\_loss / l))

# 模型持久化，将训练好的模型保存到本地

saver = tf.train.Saver(max\_to\_keep=10)

saver.save(sess, save\_path+'model', global\_step=model.max\_step)

# 填补阶段，输入预填补后的样本，计算并返回模型输出

def impute(model, test\_x, save\_path):

# 输入

# model 网络模型

# test\_x 预填补后的样本集合

# save\_path 训练后模型在本地的保存位置，例如 ‘./MODEL/’

# 输出

# outputs 网络模型的输出，用于填补缺失值

with tf.Session() as sess:

# 从本地加载训练完成的模型

saver = tf.train.Saver(max\_to\_keep=10)

saver.restore(sess, save\_path+'model-'+str(model.max\_step))

# 计算模型输出

outputs = sess.run(model.outputs, feed\_dict={model.inputs: test\_x})

return outputs

import math

# 产生不同的训练样本批次

def batch\_generator\_index(n, batch\_size): # 输入样本总数，每批样本数量

if n < batch\_size: # 如果样本总数小于每批样本数量，则直接返回所有样本的下标索引

yield np.arange(n)

else:

batch\_num = math.ceil(n/batch\_size) # 计算总的批次数量

indexes = np.arange(n) # 所有样本的下标索引

np.random.shuffle(indexes) # 打乱下标索引的顺序

expand\_num = batch\_size-n % batch\_size

if expand\_num != batch\_size: # 若最后一批的样本数不足，补充相应数量的下标索引

indexes = np.r\_[indexes, indexes[0:expand\_num]]

for i in range(batch\_num):

x = indexes[batch\_size\*i:batch\_size\*(i+1)] # 产生当前批次的样本下标索引

yield x # 返回当前批次的样本下标索引

代码清单5-2 融合式填补方案

# 当采用MVPT训练方案时，需将代码清单4-1中的define\_variables（）函数替换为以下函数

def define\_variables\_mvpt(self, data\_miss, data\_miss\_mask):

# 输入

# data\_miss: 不完整数据集，二维矩阵

# data\_miss\_mask： 由0和1构成的缺失值标记矩阵，现有值标记为1，缺失值标记为1

self.data\_miss = data\_miss

# 定义缺失值变量

self.data\_miss\_trainable = tf.Variable(self.data\_miss,

dtype=tf.float32, name='data\_miss\_trainable')

self.data\_miss\_mask = data\_miss\_mask

s = self.layer\_size[0]

# 当前批次中的样本下标索引

self.input\_index = tf.placeholder(tf.int32, shape=[None], name='inputs')

# 当前批次中样本对应的缺失值标记矩阵

self.input\_mask = tf.placeholder(tf.float32, shape=[None, s])

将当前批次中现有值定义为常量，缺失值定义为变量

temp\_tr = tf.gather(self.data\_miss\_trainable, self.input\_index)

temp\_no = tf.cast(tf.gather(self.data\_miss, self.input\_index),

tf.float32)

temp = tf.multiply(self.input\_mask, temp\_no) +

tf.multiply(1. - self.input\_mask, temp\_tr)

self.inputs = tf.reshape(temp, [-1, s])

# 将网络训练和填补过程结合，模型参数和缺失值共同完成更新

def train\_MVPT(model):

# 输入

# model 网络模型

# 输出

# imputation 填补后的数据集

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer()) # 初始化网络参数

n = len(model.data\_miss) # 数据集中的样本个数

for step in range(1, model.max\_step + 1):

tot\_loss, l = 0, 0 # tot\_loss记录每次训练时的损失值，l为累加变量

# 产生训练批次

generator = batch\_generator\_index(n, model.batch\_size)

for index in generator:

feed = {model.input\_index: index,

model.input\_mask: model.data\_miss\_mask[index, :]}

batch\_loss, \_ = sess.run([model.loss, model.optimizer],

feed\_dict=feed)

tot\_loss, l = tot\_loss + batch\_loss, l + 1

if step == 1 or step % 100 == 0: # 每训练100次，显示当前训练进度

print('step: {}/{}... '.format(step + 1, model.max\_step),

'loss: {:.8f}... '.format(tot\_loss / l))

# 填补缺失值

feed = {model.input\_index: np.arange(n),

model.input\_mask: model.data\_miss\_mask}

outputs = sess.run(model.outputs, feed\_dict=feed)

imputation = np.array(model.data\_miss)

# 寻找数据集中的缺失位置

miss\_bits = np.asarray(1 - model.data\_miss\_mask, np.bool)

imputation[miss\_bits] = outputs [miss\_bits]

return imputation # 返回填补后的数据集

代码清单6-1 基于TS建模的非线性回归填补法

load init\_Data.txt;%加载不完整数据集

data = init\_Data;

cluster\_n = #;%设置聚类个数

[N,M] = size(data);#获取样本个数和属性维数信息

#获取不完整数据集的缺失位置和个数

for i=1:1:M

nanIndex{i} = find(isnan(data(:,i)));

nanNum(i) = size(nanIndex{i},1);

end

misInfo.nanIndex = nanIndex;

misInfo.nanNum = nanNum;

[clustImpuSet,U,Aij,stepIn] = getClust(data,cluster\_n,misInfo);%聚类，输出隶属度/前件模糊集/输入变量集合

[randImpuSet] = meanImpu(data,misInfo);%均值预填补

initImpu = randImpuSet;

tssSet = initImpu;

[tssImpuSet] = tssImpu(tssSet,misInfo,Aij,stepIn);%tss填补

filename = 'impu\_Set';

saveFile(filename,tssImpuSet);%保存被填补的数据集

% my\_fcm.m：

function[center,U,obj\_fcn] = my\_fcm(data,cluster\_n,options)%采用模糊C均值对数据集data聚为cluster\_n类

%用法：

%1. [center,U,obj\_fcn] = FCMClust(Data,N\_cluster,options);

%2. [center,U,obj\_fcn] = FCMClust(Data,N\_cluster];

%输入：

%data: n\*m矩阵，表示n个样本，每个样本具有m维特征

%N\_cluster 标量，表示聚类中心数目，即类别数

%options 4\*1矩阵，其中

%options(1) 隶属矩阵U的参数，>1 (缺省值：2.0)

%options(2) 最大迭代次数 （缺省值：100）

%options(3) 隶属度最小变化量，迭代终止条件 （缺省值：1e-5）

%options(4) 每次迭代是否输出信息标志 （缺省值：1）

%输出：

%center： 聚类中心

%U： 隶属度矩阵

%obj\_fcn： 目标函数值

%判断输入参数个数，只能是2个或3个

if nargin ~=2 & nargin ~= 3

error('Too many or too few input arguments!');

end

data\_n = size(data,1);%求出data的样本个数

attr\_n = size(data,2);%求出data的属性维度

%默认操作参数

denfault\_options=[2;%隶属度矩阵U的指数

100;%最大迭代次数

1e-5;%隶属度最小变化量，迭代终止条件

1];%每次迭代是否输出信息标志

%分析无options做参数时候的情况

if nargin == 2

options = denfault\_options;

%分析有options做参数时候的情况

else

%如果用户给的options数目小于4，那么其他使用默认值

if length(options) < 4

tmp = denfault\_options;

tmp(1:length(options)) = options;

options = tmp;

end

nan\_index = find(isnan(options)==1);%将options中数的值设置为为0（NAN），不是数时设置为为1。

options(nan\_index) = denfault\_options(nan\_index);%将default\_options中对应位置的参数赋值给options中不是数的位置。

if options(1) <= 1

error('The exponent should be greater than 1!');

end

end

%将options中的分量分别复制给四个变量

expo = options(1);%隶属度矩阵U的指数

max\_iter = options(2);%最大迭代次数

min\_impro = options(3);%隶属度最小变化量，迭代终止条件

display = options(4);%每次迭代是否输出信息标志

obj\_fcn = zeros(max\_iter,1);%初始化输出参数

U = initfcm(cluster\_n,data\_n);%初始化模糊分配矩阵，使U满足列上相加为1

% main loop 主程序：获取聚类的信息

for i=1:max\_iter

%在第K步循环中改变聚类中心center和分配函数U的隶属度值

[U,center,obj\_fcn(i)] = stepfcm(data,U,cluster\_n,expo);

if display

end

%终止判断条件

if i > 1

if abs(obj\_fcn(i) - obj\_fcn(i-1)) < min\_impro

break;

end

end

end

iter\_n = i;%实际迭代次数

obj\_fcn(iter\_n+1 : max\_iter) = [];

function U = initfcm(cluster\_n,data\_n)%子函数，模糊矩阵初始化

%初始化fcm的隶属度函数矩阵

%输入：

%cluter\_n 聚类中心个数

%data\_n 样本个数

%输出：

% U 初始化的隶属度函数矩阵

U = rand(cluster\_n,data\_n);%随机初始化

U = U./(ones(cluster\_n,1)\*sum(U));%使得每列隶属度和为1

function[U\_new, center, obj\_fcn] = stepfcm(data,U,cluster\_n,expo)%子函数，模糊C均值聚类时迭代的一步

%输入：

%data n\*m矩阵，表示n个样本，每个样本具有m维特征

%U 隶属度矩阵

%cluster\_n 标量，表示聚类中心数目，即类别数

%expo 隶属度矩阵U的指数

%输出：

%U\_new 迭代计算出的新的隶属度矩阵

%center 迭代计算出的新的聚类中心

%obj\_fcn 目标函数值

%初始化

Ik = ones(size(data));

miss\_pos = find(isnan(data));

Ik(miss\_pos)=0;

data(miss\_pos)=0;

clear miss\_pos;

mf = U.^expo;%隶属度矩阵指数运算结果

center = (mf\*data)./(mf\*Ik);%新的聚类中心

dist = distfcm(center,data,Ik);%计算距离矩阵

obj\_fcn = sum(sum((dist.^2).\*mf));%计算目标函数值

tmp = dist.^(-2/(expo-1));

U\_new = tmp./(ones(cluster\_n,1)\*sum(tmp));%计算隶属度矩阵

function out = distfcm(center,data,I)%子函数，计算距离

%输入;

%center 聚类中心

%data 样本个数

%输出：

%out 距离

C = size(center,1); %获取聚类个数

[N,T] = size(data); %获取样本个数和属性维度

out = zeros(C,N); %初始化

%对每个聚类中心计算距离

for i = 1:C;

%计算每个样本到各聚类中心的距离

for j = 1:N

valid\_n = size(find(I(j,:)~=0),2);%使用局部距离

out(i,j) = T/valid\_n \* sqrt(sum(((center(i,:)-data(j,:)).^2).\*I(j,:)));

end

end

% getClust.m：

function [ impuSet, U, Aij, stepIn ] = getClust( data, cluster\_n, misInfo )% getClust 对不完整数据集聚类，并输出聚类中心和隶属度信息

nanIndex = misInfo.nanIndex;

nanNum = misInfo.nanNum;

[data\_n,attr\_n] = size(data);%获取样本个数和属性维度

Ix = 1 - isnan(data);

[center,U,obj] = my\_fcm(data,cluster\_n);

sortdata = horzcat(center,U);

%以第一列为基准从小到大排序

sortdata = sortrows(sortdata,1);

center = sortdata(:,1:attr\_n);

U = sortdata(:,attr\_n+1:end);

% 得到前提参数

preInput = data;

preInput(isnan(preInput)) = -200;

ux = U\*(preInput.\*Ix);

sum\_u = U \* Ix;

aCenter = ux ./ sum\_u;

for i=1:1:cluster\_n

var = U(i,:)\*((preInput-ones(data\_n,1)\*aCenter(i,:)).\*Ix).^2;

stdVar(i,:) = sqrt(2\*var./sum\_u(i,:));

Aij{i} = exp(-((preInput-ones(data\_n,1)\*aCenter(i,:)).\*Ix).^2 ./ (ones(data\_n,1)\*stdVar(i,:)).^2);

end

[value,label] = max(U);

% 类中心填补

impuSet = data;

for i=1:1:cluster\_n

classIndex = find(label==i);

classData = impuSet(classIndex,:);

for j=1:1:attr\_n

nanIndex = find(isnan(classData(:,j)));

classData(nanIndex,j) = center(i,j);

end

impuSet(classIndex,:) = classData;

end

% 变量选取

for i=1:1:attr\_n

in = setdiff(1:attr\_n,i);

curX = impuSet(:,in);

curY = impuSet(:,i);

%对每类数据进行变量选取

for j=1:1:cluster\_n

classIndex = find(label==j);

classX = curX(classIndex,:);%模型的输入

classY = curY(classIndex,:);模型的实际输出

[b,bint,pval,inmodel] = stepwisefit(classX,classY,'penter',0.10,'premove',0.10);%调用逐步回归算法

stepIn{i,j} = find(inmodel==1);

end

end

end

% meanImpu.m：

function [ impuSet ] = meanImpu( data,misInfo )%对不完数据集进行均值预填补以初始化缺失值

attr\_n = size(data,2);%属性个数

nanIndex = misInfo.nanIndex;%缺失值在不完整数据集中位置

impuSet = data;

%依次对每列属性进行均值预填补

for i=1:1:attr\_n

randData = zeros(size(nanIndex{i}));

impuSet(nanIndex{i},i) = randData;

meanVal = mean(impuSet(:,i));

randData =meanVal \* ones(size(nanIndex{i}));

impuSet(nanIndex{i},i) = randData;

randData=[];

end

end

% tssImpu.m:

function [ data ] = tssImpu( data, misInfo, Aij, stepIn )

%基于TS建模的非线性回归填补法

[data\_n,attr\_n] = size(data);%获取样本个数和属性维度

cluster\_n = size(Aij,2);%获取聚类个数

%获取缺失信息

nanIndex = misInfo.nanIndex;

nanNum = misInfo.nanNum;

%填补过程

for i=1:1:attr\_n

curX = data(:,setdiff(1:attr\_n,i));第i个ts模型的输入

curY = data(:,i);%第i个ts模型的实际输出

bx = [];%初始化

tt = 0;%统计规则数目

%获取每个样本在不同规则中的权重

for j=1:1:cluster\_n

%获得在每个ts模型的每个规则中显著特征的序号

curIn=stepIn{i,j};

curIn\_0=setdiff(1:attr\_n,i);

curIn\_1=curIn\_0(curIn);

if ~isempty(curIn)

tt = tt+1;

fullfill(:,tt) = min(Aij{j}(:,curIn\_1),[],2);%最小值原则求得权重

end

end

%归一化处理

sumA = sum(fullfill,2)\*ones(1,size(fullfill,2));

fullfill = fullfill ./ sumA;

tt = 0;%统计规则数目

%模型后件参数辨识以及获得模型输出

for j=1:1:cluster\_n

curIn = stepIn{i,j};%显著的输入变量序号

if ~isempty(curIn)

tt=tt+1;

stepX = curX(:,curIn);%将不显著变量剔除后的模型输入

%将所有规则的输入进行加权求和

ix = [ones(data\_n,1),stepX];

bx = [bx,fullfill(:,tt)\*ones(1,size(ix,2)).\*ix];

end

end

[b,bint,R] = regress(curY,bx);%最小二乘法求得模型参数

out(:,i) = bx \* b;%利用模型参数求取模型输出

end

%将缺失值由对应的模型输出进行替换

for i=1:1:attr\_n

data(nanIndex{i},i) = out(nanIndex{i},i);

end

end

代码清单7-1 基于交替学习策略的TS模型填补方法

function [ iterData ] = tssiImpu( data, misInfo, Aij, stepIn, iter\_n )

%tssiImpu 基于优化的TS模型的填补法

%将6-1中的tssImpu函数替换为tssiImpu函数，其他函数不变

[data\_n,attr\_n] = size(initSet);%获取样本个数和属性维度

cluster\_n = size(Aij,2);%获取聚类数目

%获取缺失信息

nanIndex = misInfo.nanIndex;

nanNum = misInfo.nanNum;

%前提参数辨识

for i=1:1:attr\_n

tt = 0; %统计规则数目

for j=1:1:cluster\_n

%获取第i个TS模型的第j个规则中显著特征的序号

curIn=stepIn{i,j};

curIn\_0=setdiff(1:attr\_n,i);

curIn\_1=curIn\_0(curIn);

if ~isempty(curIn)

tt = tt+1;

fullfill{i}(:,tt) = min(Aij{j}(:,curIn\_1),[],2);%由最小值原则取得该条规则的权重

end

end

%对每条规则的权重进行归一化处理

sumA = sum(fullfill{i},2)\*ones(1,size(fullfill{i},2));

fullfill{i} = fullfill{i} ./ sumA;

end

%基于交替学习策略的结论参数辨识以及缺失值填补

iterData = initSet;%初始化重构的完整数据集

for k=1:1:iter\_n%iter\_n为迭代次数

for i=1:1:attr\_n

curX = iterData(:,setdiff(1:attr\_n,i));%第i个TS模型的输入

curY = iterData(:,i);%第i个TS模型的输出

bx = [];%初始化

tt=0;%统计规则数目

%将每条规则的模型输出进行加权求和

for j=1:1:cluster\_n

curIn = stepIn{i,j};%第i条规则中显著输入特征的序号

if ~isempty(curIn)

tt=tt+1;

stepX = curX(:,curIn);%剔除不显著特征后的模型输入

ix = [ones(data\_n,1),stepX];

bx = [bx,fullfill{i}(:,tt)\*ones(1,size(ix,2)).\*ix];%加权后的每条规则的输入

end

end

[b,bint,R] = regress(curY,bx);%最小二乘法求取模型参数

out{k}(:,i) = bx \* b;%基于求取的模型参数获得模型输出

end

%更新填补值

for i=1:1:attr\_n

iterData(nanIndex{i},i) = out{k}(nanIndex{i},i);

end

end

end

代码清单8-1 基于聚类算法的贫困家庭类别划分

import numpy as np

import sys

def euclidean(x,y):#欧式距离，输入x，y两个向量

return np.sqrt(np.sum(np.square(x-y)))#输出两个向量间的欧式距离

def find\_martrix\_min\_value(data\_matrix):#获取矩阵的最小值，输入矩阵data\_matrix

new\_data=[]

for i in range(len(data\_matrix)):

new\_data.append(min(data\_matrix[i]))

return(min(new\_data))#输出矩阵的最小值

X = np.loadtxt('CFPS2016.txt')#载入数据集

n = X.shape[0]#样本数量

s = X.shape[1]#属性数量

d\_thr = \*#初始化簇间距距离阈值

V=X#初始化原型矩阵

C=[]

for i in range (n):

C.append(list(i))

K=n#当前簇数量

D=np.zeros([n,n])#初始化距离矩阵

#计算当前距离矩阵

for i in range(n):

for l in range(n):

D[i][l] = euclidean(X[i,:],X[l,:])

D[l][i] = D[i][l]

while find\_martrix\_min\_value(D) < d\_thr:

p = -1

q = -1

Euc\_max = sys.maxint

#获取欧式距离最近的两个簇原型，其编号分别记录为p、q

for p\_temp in range(K):

for q\_temp in range(K):

if euclidean(V[p\_temp,:],V[q\_temp,:])<Euc\_temp:

Euc\_temp = euclidean(V[p\_temp,:],V[q\_temp,:])

p = p\_temp

q = q\_temp

C[p].append(C[q])#将第q个簇合并到第p个簇

C.pop(q)#删除原第q个簇，并对第q+1、...、第K个簇重新编号为q、...、K-1

delete(D,q,axis=0)#删除距离矩阵的第q行

delete(D,q,axis=1)#删除距离矩阵的第q列

K -= 1#现有簇数量减1

#更新第p个原型

temp = np.zeros([1,s])

for count in range(len(C[p])):

temp += X[C[p][count],:]

V[p,:] = temp/len(C[p])

#更新距离矩阵D

for l in range(K):

D[p][l] = euclidean(V[p,:],V[l,:])

D[l][p] = D[p][l]

print(C)#输出聚类结果

代码清单8-2 贫困家庭典型特征分析

import numpy as np

from sklearn.linear\_model import LogisticRegression

X = np.loadtxt('CFPS2016')#载入数据集

Y = np.loadtxt('Label')#载入样本标签

s = X.shape[1]#数据集中属性数量

logistic = LogisticRegression(penalty = 'l1', C = 1)#初始化l1正则化的逻辑回归模型

logistic.fit(X,Y)#训练逻辑回归模型

coef = logistic.coef\_#获取模型权重参数

weight = np.zeros([1,s]) #初始化各特征对应权重组成的矩阵

for m in range(s):#由模型权重参数计算各特征的权重

weight[0][m] += abs(coef[0][m])

print('特征权重为：'+ weight)#输出特征权重