杭州电子科技大学计算机学院

数据仓库与数据挖掘

实验1：数据预处理

时间：2019年10月14日，学号：17061833 姓名：於文卓

注意：

1）实验开始前，填写学号和姓名。

2）将文件名由“数据预处理”更改为“学号+姓名+数据预处理”。

3）作业做完后，验收之后通过作业提交系统提交。提交网址是：

https://www.wjx.cn/jq/46478225.aspx

# 一、实验目的

1、掌握数据挖掘中数据预处理的方法；

2、了解数据转换的过程和方法；

3、了解描述性数据汇总的计算机实现方法。

# 二、实验原理

现实世界中的数据库极易受噪音数据、遗漏数据和不一致性数据的侵扰，为提高数据质量进而提高挖掘结果的质量，产生了大量数据预处理技术。数据预处理有许多方法：

(1) 数据清理: 数据清理是完成格式的标准化、对空缺值进行处理、清除重复的数据以及对异常数据进行错误纠正和清除等操作；

(2) 数据集成: 数据集成是将来自不同数据源的数据合并为统一一致的数据存储中, 这种数据存储可以是数据库或数据仓库;数据集成主要包括:包含相同字段属性的纵向追加和具有相关属性叠加的横向合并。

(3) 数据归约: 数据归约是针对原始数据集中地属性和记录, 实现有效的数据采样与对应属性选择, 进一步降低数据规模, 在数据归约过程可以采用聚集、聚类以及将冗余特征值删除等形式, 达到既能最大限度的保持数据的原有特征, 又能够有效的精简数据量的目的。数据归约主要通过数据立方体技术、维消减、数据压缩、数据块消减、离散化和概念层次生成等方法实现。

(4) 数据变换: 数据变换是根据需要将数据压缩到较小的区间中, 也就是对数据进行规格化处理, 将数据压缩到特定的范围之内。

以上几种数据预处理方法, 相互之间不仅关联而且是独立的, 各个预处理方法的实施并没有先后顺序的严格制约, 并且相互贯通, 例如消除数据冗余的过程既可以看做是数据清洗过程的一项工作, 也可以认为是数据归约工作中的一种方法。

特征选择能够从数据集中选取具有代表性的特征子集，删除不相关或冗余特征。因此，在软件缺陷预测中采用特征选择，不仅能够提高预测模型的训练速度，更重要的是能够提高其预测性能。根据特征选择的输出类型不同，可将其分为特征排序和特征子集选择两类。

# 三、实验内容：

## 实验题目

（1）实验数据：

对数据集D（CM1软件缺陷预测中常用的数据集） 进行如下特征选择处理，使用熟悉的程序设计语言进行编程（要求程序具有通用性）：

（本实验中数据集D指CM1数据集，数据细节参见CM1.arff文件）

CM1数据说明：LOC\_BLANK、 LOC\_BLANK、 BRANCH\_COUNT、CALL\_PAIRS 、LOC\_CODE\_AND\_COMMENT、LOC\_COMMENTS、CONDITION\_COUNT 、CYCLOMATIC\_COMPLEXITY、CYCLOMATIC\_DENSITY、DECISION\_COUNT、DECISION\_DENSITY 、DESIGN\_COMPLEXITY、DESIGN\_DENSITY、EDGE\_COUNT、ESSENTIAL\_COMPLEXITY、ESSENTIAL\_DENSITY、LOC\_EXECUTABLE、PARAMETER\_COUNT、HALSTEAD\_CONTENT、HALSTEAD\_DIFFICULTY、HALSTEAD\_EFFORT、HALSTEAD\_ERROR\_EST、HALSTEAD\_LENGTH、 HALSTEAD\_LEVEL、 HALSTEAD\_PROG\_TIME、 HALSTEAD\_VOLUME、 MAINTENANCE\_SEVERITY、 MODIFIED\_CONDITION\_COUNT、 MULTIPLE\_CONDITION\_COUNT、 NODE\_COUNT、 NORMALIZED\_CYLOMATIC\_COMPLEXITY、 NUM\_OPERANDS、 NUM\_OPERATORS 、 NUM\_UNIQUE\_OPERANDS、 NUM\_UNIQUE\_OPERATORS、 NUMBER\_OF\_LINES、 PERCENT\_COMMENTS 、 LOC\_TOTAL 为样本特征；

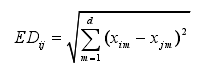
Defective {Y,N}表示样本的类别，Y表示有缺陷样本，N表示无缺陷样本。

（3）实验要求①：

基于相似性度量的特征选择方法

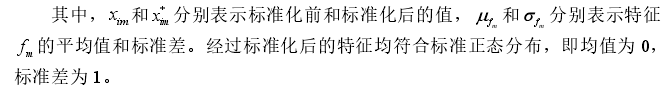
方法说明：

在软件缺陷预测中，一个数据集是由大量样本组成的，每个样本又包括多个特征来描述样本的相关特性。对于一个数据集D={x1,x2,…,xn}，该数据集包括n个样本，每个样本含有d个特征，分别表示为F={ f1,f2,…,fd }。此时，每个样本可以看作是一个 d 维向量。因此，数据集 D中任意两个样本xi与xj之间的欧氏距离为：

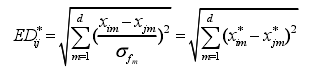


上式将数据集中不同特征的量纲(即单位)看作是相同的，故无法准确度量不同量纲的特征间的距离。因此，需要对数据集进行标准化处理，消除不同特征间的量纲约束，从而准确度量不同特征间的距离。标准化过程如下：

C:\Users\wdls\AppData\Roaming\Tencent\Users\9536317\QQ\WinTemp\RichOle\GBTA`(SKF~(A_YJ)_S6)S]C.png

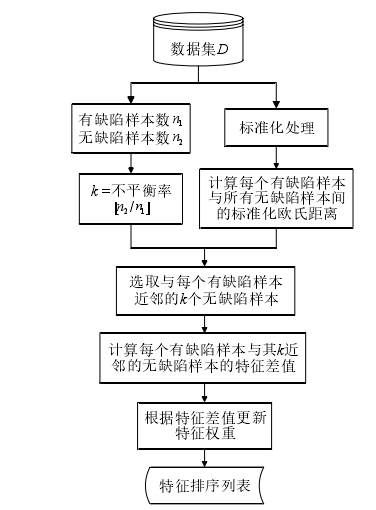


经过标准化处理后，两个样本间的标准化欧氏距离如式所示：



值越小，说明两个样本越相似。对于两个不同类别的样本来说，若它们之间的相似性较高，则认为这两个样本中特征差值较大的那些特征与类别的相关程度更高。

特征选择算法思路：



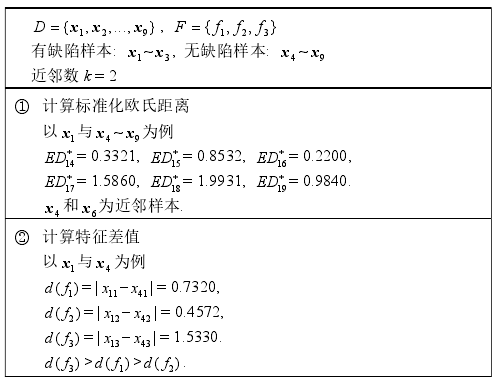
首先，统计数据集 D （即CM1）中的有缺陷样本数n 1 和无缺陷样本数n 2 ，则该数据集的不平衡率表示为⌊n2 /n1 ⌋。同时，对数据集 D 进行标准化处理，计算每个有缺陷样本与所有无缺陷样本间的标准化欧氏距离。其次，根据得到的标准化欧氏距离，选取与每个有缺陷样本最近邻的 k 个无缺陷样本，这里的k 即为数据集的不平衡率。然后，分别计算它们的特征差值，依据特征差值更新特征权重。最后，按照特征权重降序排列产生一个特征排序列表，排序越靠前，说明该特征与类别的相关程度越高。

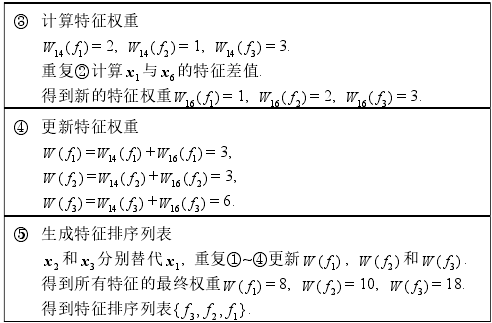
<说明：本次实验要求给特征排序列表即可>

举例说明特征选择算法过程：

假设数据集 D 中包含 9 个样本，每个样本包括 3 个特征。假设x1 ~ x 3为有缺陷样本

(共 3 个)，x4~ x9 为无缺陷样本(共 6 个)，因此近邻数 k = 2。按照算法步骤计算标准化欧式距离、特征差值和特征权重，然后更新特征权重并生成特征排序列表{ f3, f2, f1}，其中f3 与类别特征的相关度最高，而f 1 与类别的相关度最低。





在特征排序列表的基础上，按照顺序依次进行特征子集选择。例如，得到的特征排序列表为

，d表示特征个数，排序越靠前，说明该特征与类别的相关程度越高。由此，根据特征排序顺序可以得到d个特征子集，分别为，最后评价所有特征子集的分类性能。

（3）实验要求②：

度量特征之间的关联性

方法：采用非线性的对称不确定性（简称SU）来度量特征之间的关联性

SU 借助信息论中的熵, 通过衡量两个变量*X* 和*Y* 分布的差异性来度量这两个变量的关联性, 用*X* 和*Y* 的熵对它们之间的互信息进行归一化就可以计算出SU. 将特征视为一个变量, 不同实例上特征的值就是该变量的取值, 因此可以用SU 来度量两个特征之间的关联性, 其具体计算公式为



SU 的取值范围是[0*;* 1], 取值越大, 则说明两个特征之间的关联性越强. 其中*H*(*X*) 表示变量*X* (即特征) 的熵. 假设*p*(*x*) 表示*X* 取值为*x* 的先验概率, 则*H*(*X*) 的计算公式为



IG(*X|Y* ) 称为信息增益, 它表示在给定变量*Y* 的情况下, 变量*X* 的熵的减少量. 因此在通常情况下, *X* 不确定性减少的越多, 则*X* 和*Y* 的依赖性越高, 即它们之间的关联性越强, 其计算公式为



其中*H*(*X|Y* ) 衡量的是在给定变量*Y* 的情况下, 变量*X* 的熵. 假设*p*(*y*) 表示变量*Y* 取值为*y*的先验概率, *p*(*x|y*) 表示当变量*Y* 取值为*y*, 变量*X* 取值为*x* 的后验概率, 则*H*(*X|Y* ) 的计算公式为



<说明：本次实验要求 构造出矩阵M, 其中Mi，j 表示C(fi， fj )，C(fi， fj )为借助SU 计算特征fi与特征fj 之间的关联性.>

# 四、实验步骤

### 实验一:

*# -\*- coding: utf-8 -\*-*

*# @Author: TD21forever*

*# @Date: 2019-10-19 17:05:05*

*# @Last Modified by: TD21forever*

*# @Last Modified time: 2019-10-19 17:14:18*

import pandas as pd

from scipy.io import arff

import numpy as np

*# 读取文件*

data = arff.loadarff('./CM1.arff')

df = pd.DataFrame(data[0])

head\_list = df.columns.values.tolist()

*# 标准化*

data\_without\_YN = df.drop("Defective", axis=1)

data\_normalize = (data\_without\_YN - data\_without\_YN.mean()) / \

(data\_without\_YN.std())

data\_normalize['Defective'] = df.Defective

*# 分离未标准化的yes样本和no样本*

row\_yes\_data = df[df.Defective == b'Y']

row\_yes\_data = row\_yes\_data.drop("Defective", axis=1).values

row\_no\_data = df[df.Defective == b'N']

row\_no\_data = row\_no\_data.drop("Defective", axis=1).values

*# 分离标准化后的yes样本和no样本*

*# 得到缺陷样本*

yes\_samples = data\_normalize[data\_normalize.Defective == b"Y"]

yes\_samples = yes\_samples.drop("Defective", axis=1)

*# 得到无缺陷样本*

no\_samples = data\_normalize[data\_normalize.Defective == b"N"]

no\_samples = no\_samples.drop("Defective", axis=1)

*# 近邻数k*

k = len(no\_samples) // len(yes\_samples)

yes\_samples\_array = yes\_samples.values

no\_samples\_array = no\_samples.values

array = [[np.sqrt(np.sum(np.square(x - y)))

for y in no\_samples\_array]for x in yes\_samples\_array]

array = np.array(array).argsort()[:, :k]

w = {i: 0 for i in range(yes\_samples.shape[1])}

for i in range(array.shape[0]):

for j in array[i]:

ds = np.abs(row\_yes\_data[i, :] - row\_no\_data[j, :])

ds = pd.Series(ds).rank(method='min')

for index in range(len(ds)):

w[index] += ds[index]

### 实验二:

**import** **pandas** **as** **pd**

**from** **scipy.io** **import** arff

**import** **numpy** **as** **np**

**import** **time**

**import** **numba**

data = arff.loadarff('./CM1.arff')

df = pd.DataFrame(data[0]).drop("Defective", axis=1)

head\_list = df.columns

**def** data\_discretize(dataset):

n = 20

columns = dataset.shape[1]

**for** column **in** range(columns):

*#每一个特征*

x = dataset.iloc[:,column]

y = pd.cut(x,n,labels=range(n))

dataset.iloc[:,column] = y.astype('float64')

**return** dataset

**def** get\_px(feature):

**return** feature.value\_counts() / df.shape[0]

*# 传入一个特征*

**def** get\_hx(feature):

px = get\_px(feature)

log2px = np.log2(px)

Hx = -np.sum(log2px \* px)

**return** Hx

**def** get\_hxy(X, Y):

pxy = pd.crosstab(X, Y, margins=True).apply(**lambda** x: x / x[-1])

log2pxy = np.log2(pxy)

where\_is\_inf = np.isinf(log2pxy)

log2pxy[where\_is\_inf] = 0.0

py = get\_px(Y)

hxy = -np.sum(py \* np.sum(pxy \* log2pxy))

**return** hxy

**def** get\_su(X, Y):

hx = get\_hx(X)

hy = get\_hx(Y)

hxy = get\_hxy(X, Y)

igxy = hx - hxy

suxy = 2 \* igxy / (hx + hy)

**return** suxy

*#离散化 20组*

df = data\_discretize(df)

M = np.zeros((df.shape[1], df.shape[1]))

**for** featurex **in** range(df.shape[1]):

**for** featurey **in** range(df.shape[1]):

res = get\_su(df.iloc[:, featurex], df.iloc[:, featurey])

M[featurex][featurey] = res

**print**("M[{}][{}]={}".format(featurex, featurey, res))

M = pd.DataFrame(M)

M.index = head\_list

M.columns = head\_list

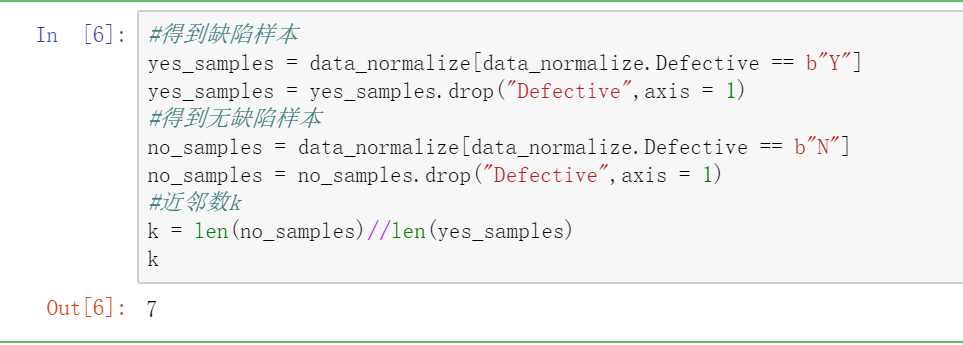
**print**(M)

# 五、实验结果

(截图，分析说明)

### 实验一:

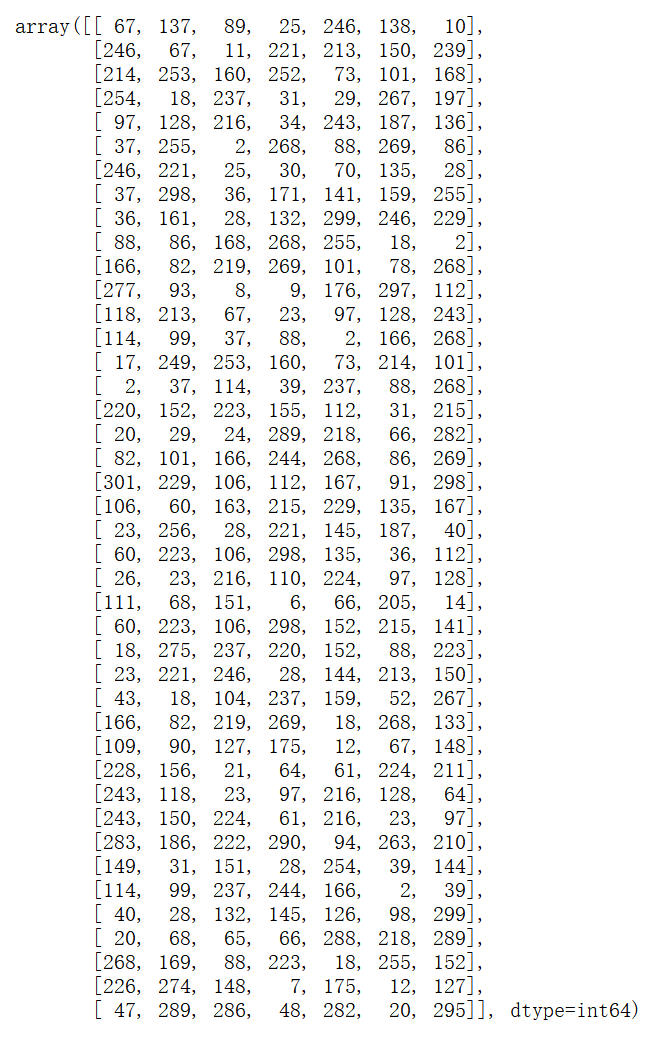
1. 首先得到不平衡率K为7



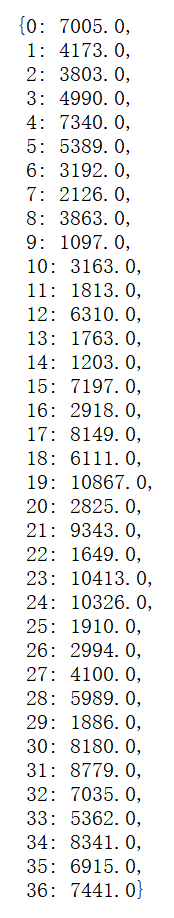
1. 对数据标准化



1. 每个有缺陷样本与所有无缺陷样样本标准化欧氏距离,选取邻近的k个

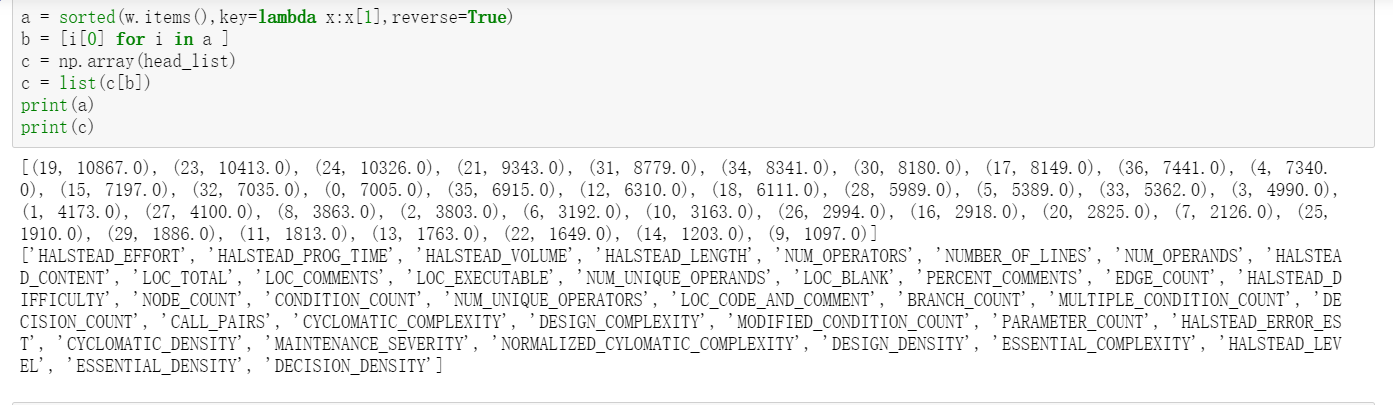


1. 特征差值,并更新权重



其中0---36表示f1到f37这37个特征,对应的值表示其权重

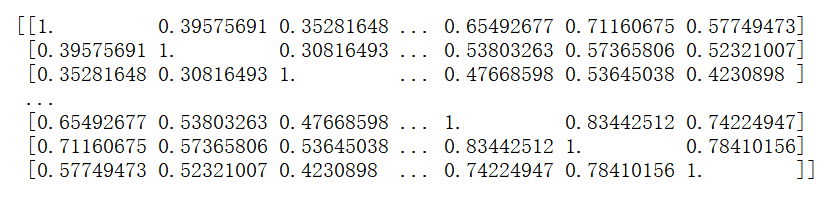
1. 特征排序

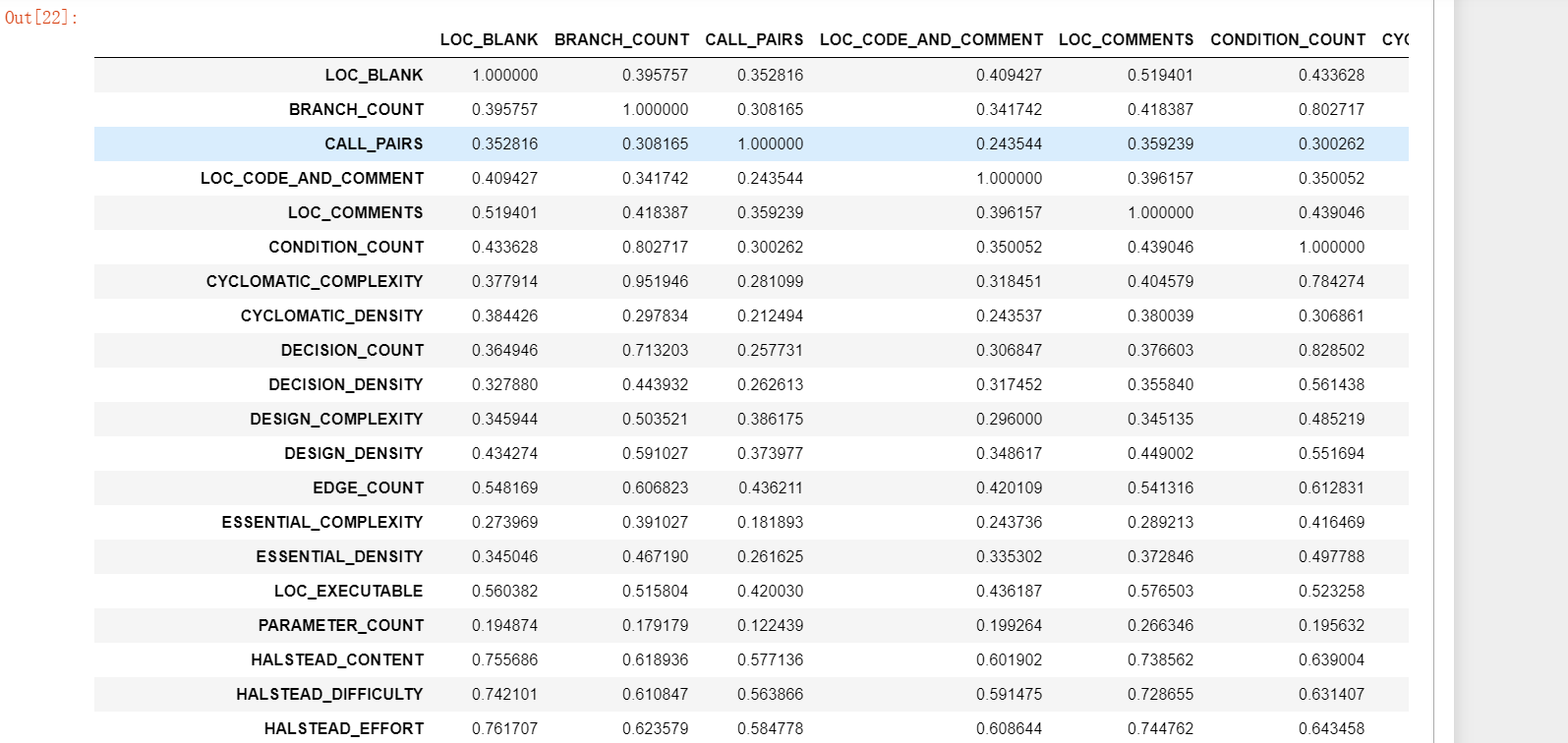


### 实验二

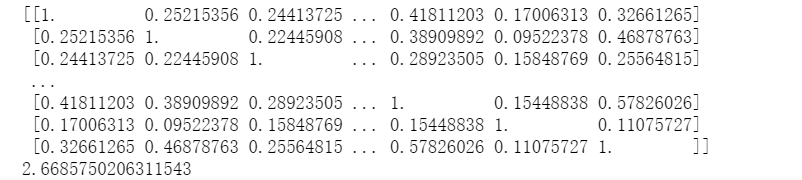
#### 未离散化结果

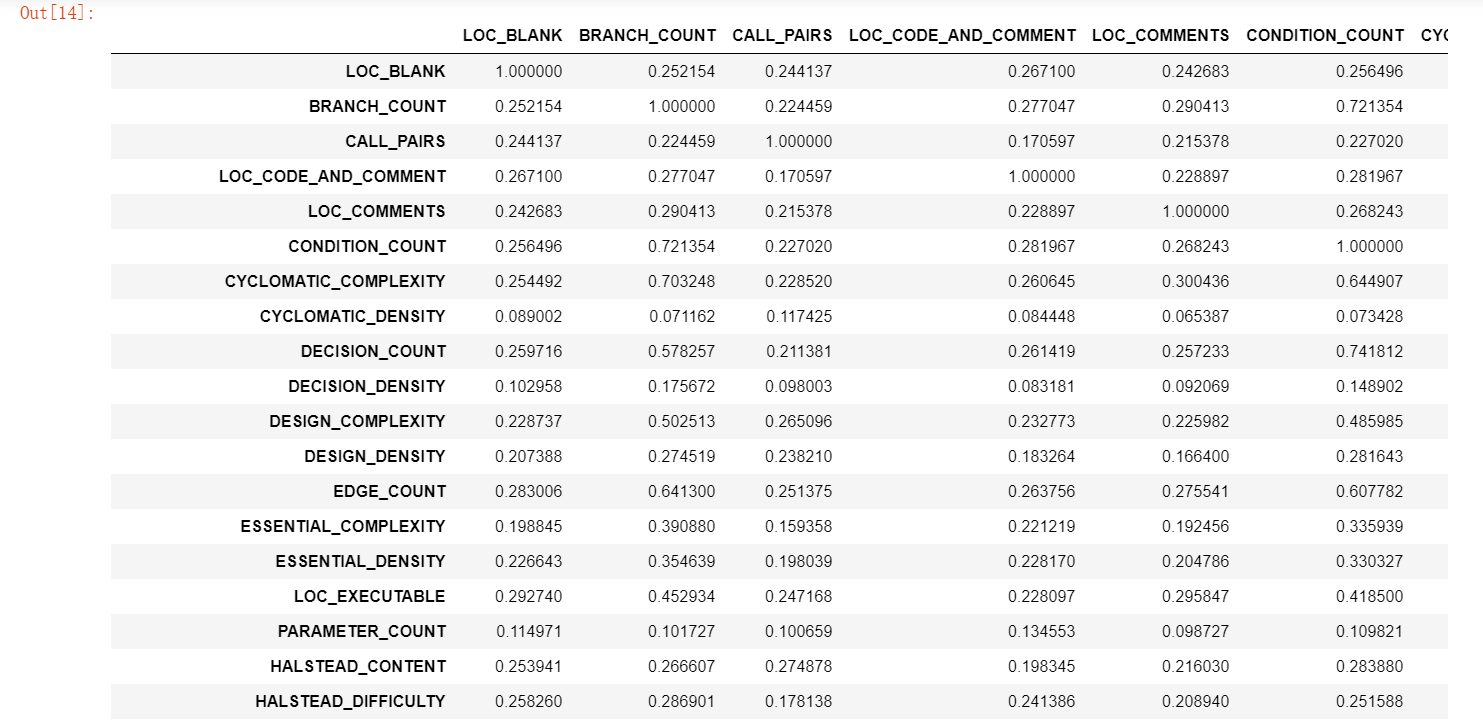
矩阵M





#### 20组离散化结果





# 六、心得体会

1. 最大的感受就是这种理论性这么强的方法,必须通过实践应用才能理解那些数学公式的真正含义,也能理解这些算法的应用场景

2. 在实验过程中,最好使用小数据来检测自己代码是否写的正确.比如实验一,使用老师提供的小数据,实验二选择第一个特征和第二个特征,带入已经写好的代码中,来检测其正确性.如果没问题,再修改代码扩大数据量,使之适合整个数据集

3. 学会使用工具和库,比如实验二要求P(X|Y),python的pandas中的crosstab函数很好的完成了这个要求,rank()函数也能快速排出一组数中每个元素的排名,有利于使得代码更加易读,逻辑更加清楚

4. 本次实验帮助我了解了数据预处理的一些方法的同时,巩固了python处理数据中,numpy和pandas两个库的使用