

电子科技大学

计算机专业类课程

实验报告

课程名称：数据挖掘与大数据分析

学 院：计算机科学与工程学院

专 业：计算机科学与技术

学生姓名：岳子豪

学 号：2018051404015

指导教师：邵俊明

日 期： 2021 年 9 月 11 日

电子科技大学

实验报告

实验一

一、实验名称：数据预处理

二、实验学时：4

三、实验内容和目的：

1. 安装并配置 Python、Pycharm 和 Weka
2. 使用图形界面的 Weka 工具包，完成数据归一化、缺失值处理、特征筛选的数据预处理操作
3. 在 Pycharm 下调用 numpy、pandas 和 Counter 包，完成数据归一化、缺失值处理、特征筛选的数据预处理操作

四、实验原理：

1. 数据属性最小最大归一化。

$$v' = \frac{v - \min_A}{\max_A - \min_A} (\text{new_max}_A - \text{new_min}_A) + \text{new_min}_A$$

其中 v 是属性 A 的某个观测值， \min_A 和 \max_A 分别是属性 A 的最小值和最大值。上述公式将 A 属性的取值映射到区间 $[\text{new_min}_A, \text{new_max}_A]$ ，如果令 $\text{new_max}_A = 1, \text{new_min}_A = 0$ ，则将 A 属性映射到区间 $[0,1]$ ，实现了数据归一化。

2. 缺失值处理。对于数据中属性的缺失值，使用该属性的平均值来填补缺失值。
3. 特征筛选。信息增益是用来进行特征筛选的常用算法，基本思想是选择那些特征对分类变量 Y 信息增益大，删除那些对分类无用的特征。

五、实验器材（设备、元器件）

1. 笔记本 1 台，系统 Windows 11
2. Python 3.9、Pycharm 2021.2 、Weka 3.8.5

六、实验步骤：

1. 使用 Weka 对 iris.arff 进行归一化处理，filter 使用 Normalization，参数默认。
2. 新建 experiment1_normalization.py 并编写 Python 代码，如下：

```
import numpy as np
import pandas as pd

def loadIris(address):
    spf = pd.read_csv(address, sep=',', index_col=False, header=None)
    strs = spf[4]
    spf.drop([4], axis=1, inplace=True)
    return spf.values, strs

def normalization(data_matrix):
    e = 1e-5
    for c in range(4):
        maxNum = np.max(data_matrix[:,c])
        minNum = np.min(data_matrix[:,c])
        data_matrix[:,c] = (data_matrix[:,c] - minNum + e)/(maxNum - minNum + e)
    return data_matrix

if __name__ == '__main__':
    filepath = 'iris.arff'
    writepath = 'iris_normal.txt'
    # read data
    data_matrix, str_name = loadIris(filepath)
    # normalization
    data_matrix = normalization(data_matrix)
```

```

spf = pd.DataFrame(data_matrix)
strs = str_name.values
spf.insert(4,4,strs)
spf.to_csv(writepath,index=False,header=False)

```

运行该程序，对 iris.arff 进行处理，程序将结果写入 iris_normal.txt。

再使用图形界面的 Weka 工具包对 iris.arff 进行处理，将处理后的结果保存在 iris_normal(GUI).arff 中，对比 iris_normal.txt 与 iris_normal(GUI).arff 中的内容，比较通过两种不同方法对数据进行归一化操作后的结果。

3. 使用 Weka 图形界面工具完成数据缺失值处理，数据使用 labor.arff，fil 使用 ReplaceMissingValues，参数默认。

4. 新建文件缺失值处理.py，并编写 Python 代码，如下：

```

import numpy as np
import pandas as pd
from collections import Counter

def loadLabor(address):
    spf = pd.read_csv(address, sep=',', index_col=False, header=None)
    column=['duration', 'wage-increase-first-year', 'wage-increase-second-year',
'wage-increase-third-year', 'cost-of-living-adjustment', 'working-hours', 'pension',
'standby-pay',
'shift-differential', 'education-allowance', 'statutory-holidays', 'vaction', 'longterm-
disability-assistance', 'contribution-to-dental-plan',
'bereavement-assistance', 'contribution-to-health-plan', 'class']
    spf.columns = column
    # label data
    str_typeName = ['cost-of-living-adjustment', 'pension', 'education-allowance', '
vacation', 'longterm-disability-assistance', 'contribution-to-dental-plan', '
bereavement-assistance', 'contribution-to-health-plan', 'class']
    str2numeric = {}
    str2numeric['?'] = '-1'
    spf = spf.replace(str2numeric)
    return spf, str2numeric, str_typeName

def fillMissData(spf, str2numeric):
    row, col = spf.shape
    columns = spf.columns
    for column_name in columns:

```

```

if column_name not in str2numeric:
    # number, "first stretegy"
    tmp = spf[column_name].apply(float)
    ave = np.average(tmp[tmp != -1])
    tmp[tmp == -1] = ave
    spf[column_name] = tmp
else:
    # Label, second stretegy
    v = spf[column_name].values
    v1 = v[v != '-1']
    c = Counter(v1)
    cc = c.most_common(1)
    v[v == '-1'] = cc[0][0]
return spf

if __name__ == '__main__':
    filepath = 'labor.arff'
    fillFilePath = 'laborMissing_handle.txt'
    spf, str2numeric, str2numeric = loadLabor(filepath) # Load data
    spf = fillMissData(spf, str2numeric) # fill missing data
    spf.to_csv(fillFilePath, index=False, header = False) # save data

```

运行该程序，将结果写入 laborMissing_handle.txt。

- 使用 Weka 图形界面工具完成特征筛选，数据使用 iris.arff，filter 使用 AttributeSelection，其中参数 evaluator 选择 InfoGainAttributeEval，search 使用 Ranker，调节 Ranker 的参数为 4。
- 新建文件特征筛选.py，并编写 Python 代码如下：

```

import numpy as np
import pandas as pd
from collections import Counter

def loadIris(address):
    spf = pd.read_csv(address, sep=',', index_col=False, header=None)
    strs = spf[4]
    spf.drop([4], axis=1, inplace=True)
    return spf.values, strs

def featureSelection ( features, label ) :
    featureLen = len( features[0, : ] )

```

```

label_count = Counter(label)
samples_energy = 0.0
data_len = len(label)
for i in label_count.keys() :
    label_count[i] /= float(data_len)
samples_energy -= label_count[i] * np.log2(label_count[i])
informationGain = []

for f in range(featureLen): # computing energy for each fea#discretize: 10
    af = features [ :, f]
    minf = np.min(af)
    maxf = np.max(af) + 1e-4
    width = (maxf - minf) /10.0
    d = (af - minf) / width# dividing data
    dd = np.floor(d)
    c = Counter(dd)
    sub_energy = getEnergy(c,dd,label)
    informationGain.append( samples_energy - sub_energy)
return informationGain

def getEnergy(c, data,label):
    dataLen = len (label)
    energy = 0.0
    #dataLen = Len ( data)
    for key, value in c.items ( ):
        c[key] /=float(dataLen)
        label_picked = label[data == key]
        l = Counter(label_picked)
        e = 0.0
        for k, v in l.items( ):
            r = v/float(value)
            e -= r* np.log2(r)
        energy += c[key] * e
    return energy

if __name__ == '__main__':
    filepath = 'iris.arff'
    #read data
    data_matrix, str_name = loadIris(filepath)
    informationGain = featureSelection(data_matrix, str_name.values)
    print( informationGain)

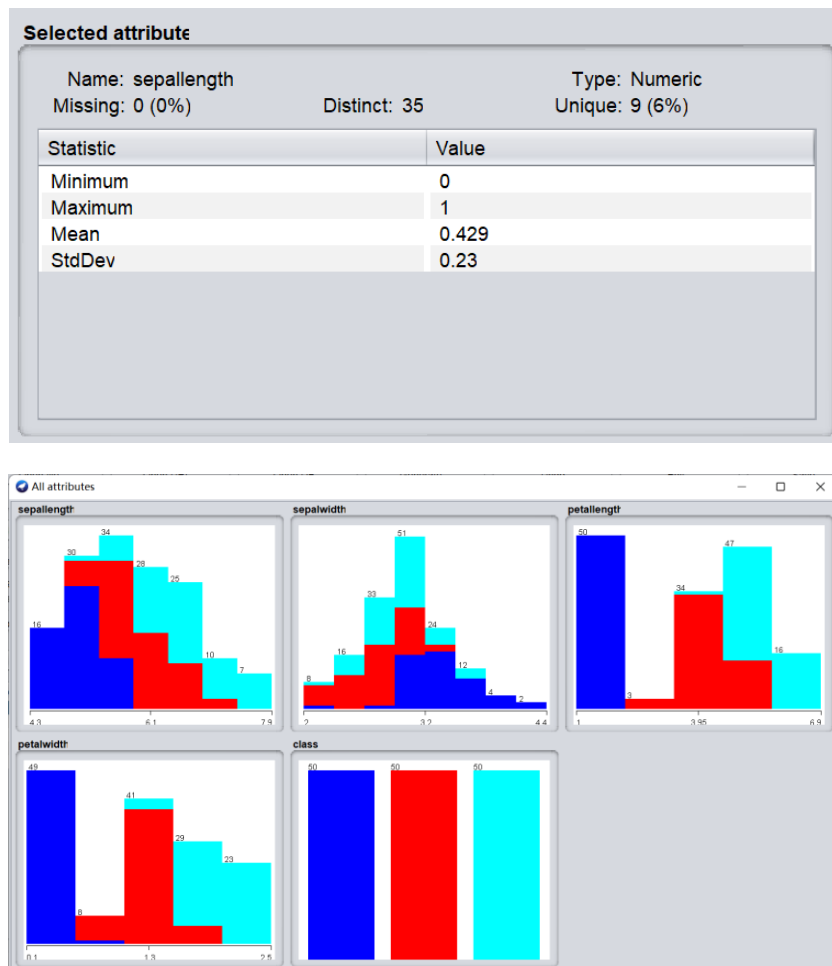
```

7. 运行该程序，观察运行结果。

七、实验数据及结果分析：

1. 归一化

使用 Weka 工具包对数据进行归一化处理，结果如下：



可以看出，数据各个属性的值已经归一化到 $[0, 1]$ 。

使用 Python 对数据进行归一化之后的结果与使用 Weka GUI 工具包处理后的结果对比如下：

实验一

| sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | class |
|------------------------|--------------------|------------------------|-----------------------|-------------|
| 0.22222438271004794 | 0.6250015624934895 | 0.06779819017255902 | 0.04167065970558456 | Iris-setosa |
| 0.166668981475805158 | 0.4166690972120949 | 0.06779819017255902 | 0.04167065970558456 | Iris-setosa |
| 0.11111380024005497 | 0.5000020833244528 | 0.050849066357514655 | 0.04167065970558456 | Iris-setosa |
| 0.08333587962255655 | 0.4583359026837384 | 0.0847473139876034 | 0.04167065970558456 | Iris-setosa |
| 0.19444668209254976 | 0.666668055497685 | 0.06779819017255902 | 0.04167065970558456 | Iris-setosa |
| 0.305557484562543 | 0.7916675347186052 | 0.11864556161769216 | 0.12500364581814244 | Iris-setosa |
| 0.08333587962255655 | 0.5833350694372106 | 0.06779819017255902 | 0.08333715276186349 | Iris-setosa |
| 0.19444668209254976 | 0.5833350694372106 | 0.0847473139876034 | 0.04167065970558456 | Iris-setosa |
| 0.0277780478387560176 | 0.3750026041558159 | 0.06779819017255902 | 0.04167065970558456 | Iris-setosa |
| 0.166668981475805158 | 0.4583359026837384 | 0.0847473139876034 | 0.166649305627893e-06 | Iris-setosa |
| 0.305557484562543 | 0.7083345486060475 | 0.0847473139876034 | 0.04167065970558456 | Iris-setosa |
| 0.13889128885755315 | 0.5833350694372106 | 0.1016964378026478 | 0.04167065970558456 | Iris-setosa |
| 0.13889128885755315 | 0.4166690972120949 | 0.06779819017255902 | 0.166649305627893e-06 | Iris-setosa |
| 2.7777700617498283e-06 | 0.4166690972120949 | 0.0169508187274259 | 0.166649305627893e-06 | Iris-setosa |
| 0.41666828703253594 | 0.8333340277748842 | 0.03389994254247026 | 0.04167065970558456 | Iris-setosa |
| 0.38889058641503776 | 1.0 | 0.0847473139876034 | 0.12500364581814244 | Iris-setosa |
| 0.305557484562543 | 0.7916675347186052 | 0.050849066357514655 | 0.12500364581814244 | Iris-setosa |
| 0.22222438271004794 | 0.6250015624934895 | 0.06779819017255902 | 0.08333715276186349 | Iris-setosa |
| 0.38889058641503776 | 0.7500010416623262 | 0.11864556161769216 | 0.08333715276186349 | Iris-setosa |
| 0.22222438271004794 | 0.7500010416623262 | 0.0847473139876034 | 0.08333715276186349 | Iris-setosa |
| 0.305557484562543 | 0.5833350694372106 | 0.11864556161769216 | 0.04167065970558456 | Iris-setosa |
| 0.22222438271004794 | 0.7083345486060475 | 0.0847473139876034 | 0.12500364581814244 | Iris-setosa |
| 0.08333587962255655 | 0.666668055497685 | 0.6949123815044382e-06 | 0.04167065970558456 | Iris-setosa |
| 0.22222438271004794 | 0.5416685763809316 | 0.11864556161769216 | 0.16667013887442136 | Iris-setosa |
| 0.13889128885755315 | 0.5833350694372106 | 0.1525438092477809 | 0.04167065970558456 | Iris-setosa |
| 0.19444668209254976 | 0.4166690972120949 | 0.1016964378026478 | 0.04167065970558456 | Iris-setosa |
| 0.19444668209254976 | 0.5833350694372106 | 0.1016964378026478 | 0.12500364581814244 | Iris-setosa |
| 0.25000208332754636 | 0.6250015624934895 | 0.0847473139876034 | 0.04167065970558456 | Iris-setosa |
| 0.25000208332754636 | 0.5833350694372106 | 0.06779819017255902 | 0.04167065970558456 | Iris-setosa |
| 0.11111380024005497 | 0.5000020833244528 | 0.1016964378026478 | 0.04167065970558456 | Iris-setosa |
| 0.13889128885755315 | 0.4583359026837384 | 0.1016964378026478 | 0.04167065970558456 | Iris-setosa |
| 0.305557484562543 | 0.5833350694372106 | 0.0847473139876034 | 0.12500364581814244 | Iris-setosa |
| 0.25000208332754636 | 0.875000520831163 | 0.0847473139876034 | 0.166649305627893e-06 | Iris-setosa |
| 0.33333587962255655 | 0.0416667 | 0.0677977 | 0.041667 | Iris-setosa |

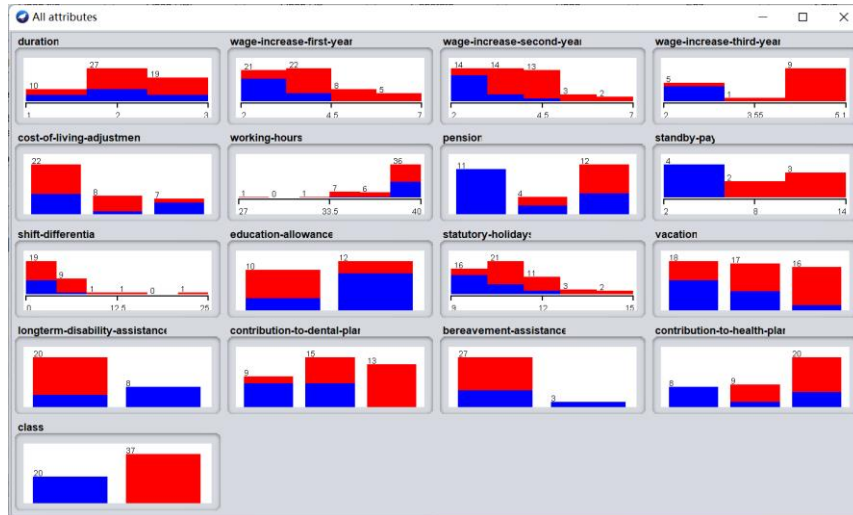
可见，在一定精度范围内，两种方法对数据进行归一化预处理的结果相同。

2. 缺失值处理

使用 Weka 对 labor.arff 进行缺失值处理，ReplaceMissingvalues 之后的结果如下：

| Selected attribute | |
|--------------------|----------------|
| Name: duration | Type: Numeric |
| Missing: 1 (2%) | Distinct: 3 |
| | Unique: 0 (0%) |
| Statistic | Value |
| Minimum | 1 |
| Maximum | 3 |
| Mean | 2.161 |
| StdDev | 0.708 |

实验一



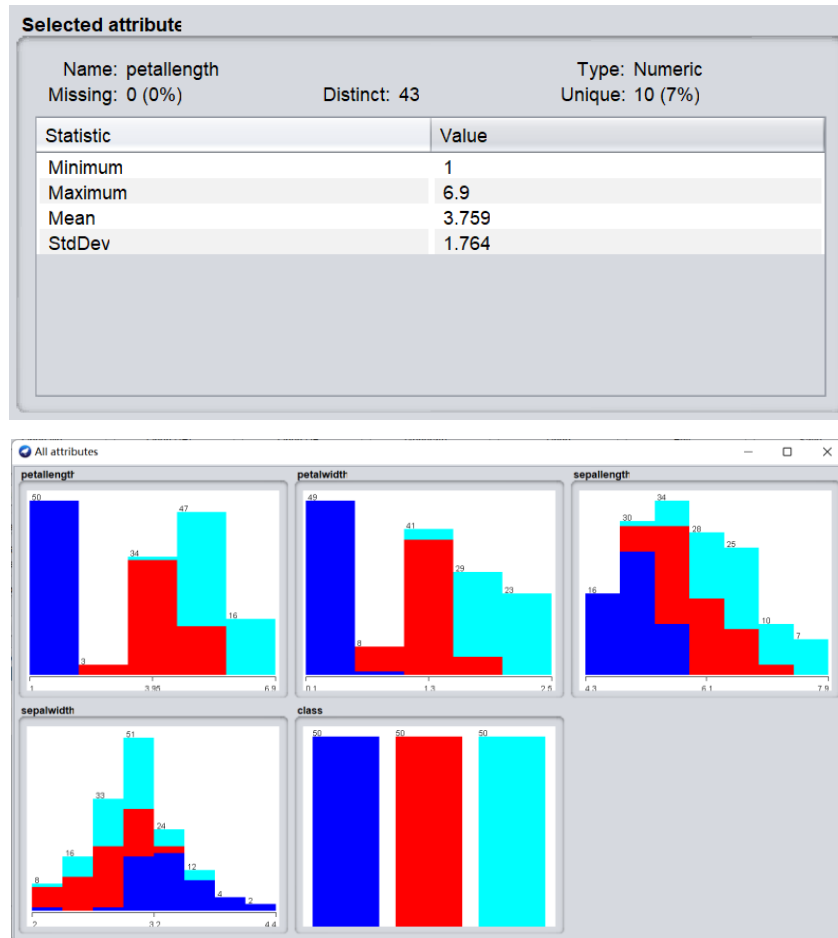
观察处理前后的数据，可以看出，数据中属性的缺失值，已经被该属性的平均值填补。

| labor.arff | laborMissing_handle.txt |
|--|--|
| 1 1,5,2,7,7,40,7,2,2,11,'average',2,2,'yes',2,'good' | 1 1,0,5,-1,-1,-1,40,-1,-1,2,-1,11,'average',-1,-1,'yes',-1,'good' |
| 2 2,4,5,5,8,7,35,'ret_allm',2,2,'yes',11,'below_average',2,'full',2,'full','good' | 2 2,0,4,5,5,8,-1,-1,35,'ret_allm',-1,-1,'yes',11,'below_average',-1,'full',-1,'full','good' |
| 3 7,7,7,7,38,'empl_contr',2,5,7,11,'generous','yes','half','yes','half','good' | 3 2,1607142857142856,-1,-1,-1,-1,38,'empl_contr',-1,5,-1,11,'generous','yes','half','yes','half' |
| 4 3,3,7,4,5,'tc',2,2,2,2,'yes',2,2,2,'yes',2,'good' | 4 3,0,3,7,4,5,'tc',-1,-1,-1,-1,'yes',-1,-1,-1,'yes',-1,'good' |
| 5 3,4,5,4,5,5,7,40,7,2,2,12,'average',2,'half','yes','half','good' | 5 3,0,4,5,4,5,5,-1,40,-1,-1,-1,-1,12,'average',-1,'half','yes','half','good' |
| 6 2,2,2,5,7,35,7,2,6,'yes',12,'average',2,2,2,'good' | 6 2,0,2,2,5,-1,-1,35,-1,6,'yes',12,'average',-1,-1,-1,-1,'good' |
| 7 3,4,5,5,'tc',2,'empl_contr',2,2,2,12,'generous','yes','none','yes','half','good' | 7 3,0,4,5,5,'tc',-1,'empl_contr',-1,-1,-1,12,'generous','yes','none','yes','half','good' |
| 8 3,6,9,4,8,2,3,7,40,7,2,3,12,'below_average',2,2,2,'good' | 8 3,0,6,9,4,8,2,3,-1,40,-1,-1,3,-1,12,'below_average',-1,-1,-1,-1,'good' |
| 9 2,3,7,7,38,7,12,25,'yes',11,'below_average','yes','half','yes',2,'good' | 9 2,0,3,7,-1,-1,38,-1,12,25,'yes',11,'below_average','yes','half','yes',-1,'good' |
| 10 1,5,7,2,2,'none',40,'empl_contr',2,4,2,11,'generous','yes','full',2,2,'good' | 10 1,0,5,7,-1,-1,'none',40,'empl_contr',-1,4,-1,11,'generous','yes','full',-1,-1,'good' |
| 11 3,3,5,4,4,6,'none',36,2,2,2,13,'generous',2,2,'yes','full','good' | 11 3,0,3,5,4,4,6,'none',36,-1,3,-1,13,'generous',-1,-1,'yes','full','good' |
| 12 2,6,4,6,4,2,38,2,2,2,15,2,2,'full',2,2,'good' | 12 2,0,6,4,6,4,-1,-1,38,-1,-1,4,-1,15,-1,-1,'full',-1,-1,'good' |
| 13 2,3,5,4,2,'none',40,2,2,2,2,'no',10,'below_average','no','half',2,'half','bad' | 13 2,0,3,5,4,-1,'none',40,-1,-1,2,'no',10,'below_average','no','half',-1,'half','bad' |
| 14 3,3,5,4,5,1,'tc',37,2,2,2,4,13,'generous',2,'full','yes','full','good' | 14 3,0,3,5,4,5,1,'tc',37,-1,-1,4,-1,13,'generous',-1,'full','yes','full','good' |
| 15 1,3,2,2,'none',36,2,2,2,10,'no',11,'generous',2,2,2,'good' | 15 1,0,3,-1,-1,'none',36,-1,-1,10,'no',11,'generous',-1,-1,-1,-1,'good' |
| 16 2,4,5,4,2,'none',37,'empl_contr',2,2,2,11,'average',2,'full','yes',2,'good' | 16 2,0,4,5,4,-1,'none',37,'empl_contr',-1,-1,-1,11,'average',-1,'full','yes',-1,'good' |
| 17 1,2,8,7,7,35,7,2,2,12,'below_average',2,2,2,'good' | 17 1,0,2,8,-1,-1,-1,35,-1,-1,2,-1,12,'below_average',-1,-1,-1,-1,'good' |
| 18 1,2,1,2,2,'tc',40,'ret_allm',2,3,'no',9,'below_average','yes','half',2,'none','bad' | 18 1,0,2,1,-1,-1,'tc',40,'ret_allm',2,3,'no',9,'below_average','yes','half',-1,'none','bad' |
| 19 1,2,2,2,'none',38,'none',2,2,'yes',11,'average','no','none','no','none','bad' | 19 1,0,2,-1,-1,'none',38,'none',-1,-1,'yes',11,'average','no','none','no','none','bad' |
| 20 2,4,5,2,'tc',35,2,13,5,2,15,'generous',2,2,2,'good' | 20 2,0,4,5,-1,'tc',35,-1,13,5,-1,15,'generous',-1,-1,-1,-1,'good' |
| 21 2,4,3,4,4,2,38,2,2,2,12,'generous',2,'full',2,'full','good' | 21 2,0,4,3,4,4,-1,-1,38,-1,-1,4,-1,12,'generous',-1,'full',-1,'full','good' |
| 22 2,2,5,3,7,40,'none',2,2,2,11,'below_average',2,2,2,'bad' | 22 2,0,2,5,3,-1,-1,40,'none',-1,-1,-1,11,'below_average',-1,-1,-1,-1,'bad' |
| 23 3,3,5,4,4,6,'tc',27,2,2,2,7,2,2,2,'good' | 23 3,0,3,5,4,4,6,'tc',27,-1,-1,-1,-1,-1,-1,-1,-1,'good' |
| 24 2,4,5,4,7,40,2,2,2,4,10,'generous',2,'half',2,'full','good' | 24 2,0,4,5,4,-1,-1,40,-1,-1,4,-1,10,'generous',-1,'half',-1,'full','good' |
| 25 1,6,7,2,38,2,8,3,2,9,'generous',2,2,2,'good' | 25 1,0,6,-1,-1,-1,38,-1,8,3,-1,9,'generous',-1,-1,-1,-1,'good' |
| 26 3,2,2,2,'none',40,'none',2,2,2,10,'below_average',2,'half','yes','full','bad' | 26 3,0,2,2,2,'none',40,'none',-1,-1,-1,10,'below_average',-1,'half','yes','full','bad' |
| 27 2,4,5,4,5,2,'tc',2,2,2,2,'yes',10,'below_average','yes','none',2,'half','good' | 27 2,0,4,5,4,5,-1,'tc',-1,-1,-1,-1,'yes',10,'below_average','yes','none',-1,'half','good' |
| 28 2,3,3,2,'none',33,2,2,2,'yes',12,'generous',2,2,'yes','full','good' | 28 2,0,3,3,-1,'none',33,-1,-1,-1,'yes',12,'generous',-1,-1,'yes','full','good' |
| 29 2,5,4,2,'none',37,2,2,5,'no',11,'below_average','yes','full','yes','full','good' | 29 2,0,5,4,-1,'none',37,-1,-1,5,'no',11,'below_average','yes','full','yes','full','good' |
| 30 3,2,2,5,2,35,'none',2,2,2,10,'average',2,2,'yes','full','bad' | 30 3,0,2,2,5,-1,-1,35,'none',-1,-1,-1,10,'average',-1,-1,'yes','full','bad' |
| 31 3,4,5,4,5,5,'none',40,2,2,2,'no',11,'average',2,'half',2,2,'good' | 31 3,0,4,5,4,5,5,'none',40,-1,-1,-1,'no',11,'average',-1,'half',-1,-1,'good' |
| 32 3,3,2,5,'tc',40,'none',2,5,'no',10,'below_average','yes','half','yes','full','bad' | 32 3,0,3,2,5,'tc',40,'none',-1,5,'no',10,'below_average','yes','half','yes','full','bad' |
| 33 2,2,5,2,5,7,38,'empl_contr',2,2,2,10,'average',2,2,2,'bad' | 33 2,0,2,5,2,5,-1,-1,38,'empl_contr',-1,-1,-1,10,'average',-1,-1,-1,-1,'bad' |
| 34 2,4,5,2,'none',40,'none',2,3,'no',10,'below_average','no','none',2,'none','bad' | 34 2,0,4,5,-1,'none',40,'none',-1,3,'no',10,'below_average','no','none',-1,'none','bad' |
| 35 3,2,2,5,2,1,'tc',40,'none',2,1,'no',10,'below_average','no','half','yes','full','bad' | 35 3,0,2,2,5,2,1,'tc',40,'none',2,1,'no',10,'below_average','no','half','yes','full','bad' |
| 36 2,2,2,2,'none',40,'none',2,2,'no',11,'average','yes','none','yes','full','bad' | 36 2,0,2,2,-1,'none',40,'none',-1,-1,'no',11,'average','yes','none','yes','full','bad' |

3. 特征筛选

使用 Weka 对 iris.arff 进行特征筛选，将最大特征数目设定为 4，结果如下：

实验一



Python 程序运行结果如下：

```
特征筛选 ×
D:\Miniconda3\envs\d2l\python.exe D:\Desktop\数据挖掘实验\DM-Experiments\DM-Exp1\特征筛选.py
[-0.3267378619587903, -0.6261502100123909, 0.29438650728661453, 0.3707140970540807]
Process finished with exit code 0
```

八、实验结论、心得体会和改进建议：

1. 实验结论：

本实验通过 python 调用 numpy、pandas、Counter 包，完成了数据归一化、缺失值处理、特征筛选等数据预处理操作。使用 Python 预处理的数据与使用图形界面的 Weka 工具包处理后得到的结果一致。

2. 心得体会：

本实验帮助我对数据预处理的基本知识和意义有了更深刻的认识，通过实践，对数据预处理的操作有了基本的掌握，为后续的实验打好了基础。

3. 改进建议：

无。

电子科技大学

实验报告

实验二

一、实验名称：关联数据挖掘

二、实验学时：4

三、实验内容和目的：

1. 掌握关联规则挖掘的基本概念、原理和一般方法
2. 掌握 Apriori 算法

四、实验原理：

1. 挖掘关联规则一般步骤

(1) 频繁项集产生 (Frequent Itemset Generation)

其目标是发现满足最小支持度阈值的所有项集，这些项集称作频繁项集。

(2) 规则的产生 (Rule Generation)

其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则 (strong rule)。

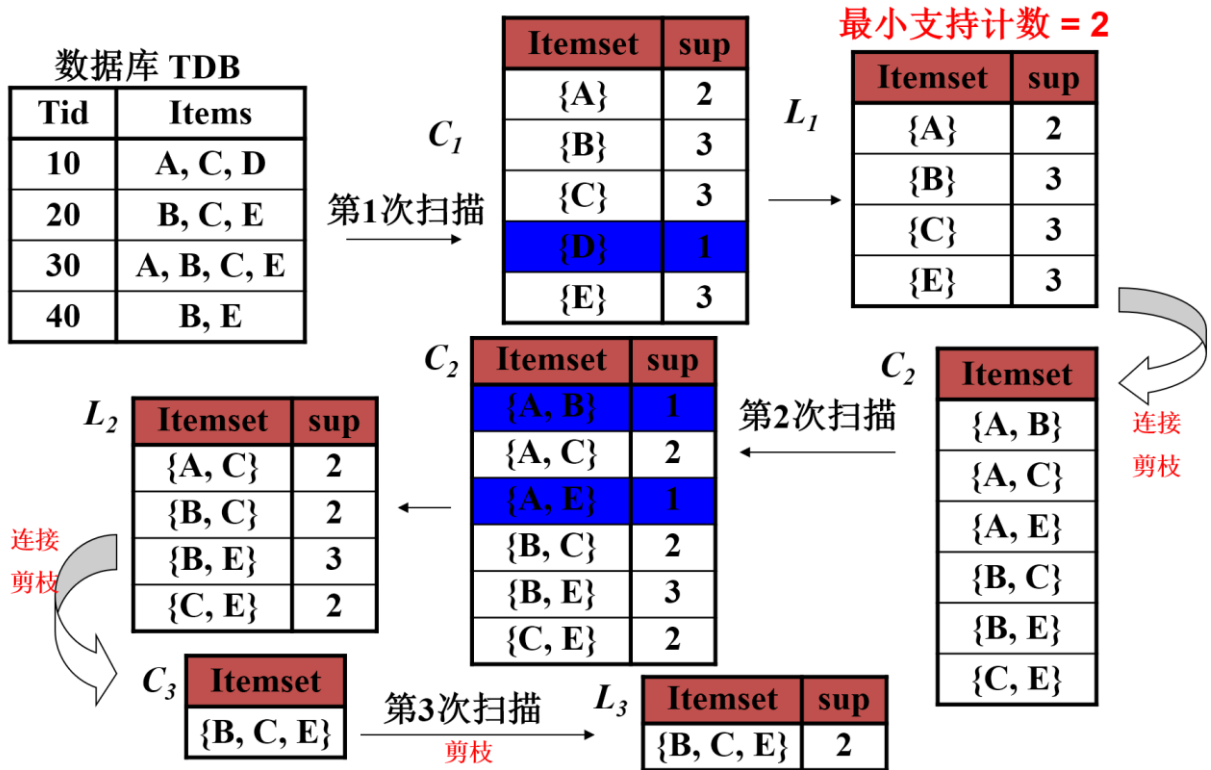
2. Apriori 算法

性质一：如果一个项集是频繁的，则它的所有子集一定也是频繁的

性质二：相反，如果一个项集是非频繁的，则它的所有超集也一定是非频

繁的

3. Apriori 算法



五、实验器材（设备、元器件）

笔记本 1 台，系统 Windows 11

Python 3.9、Pycharm 2021.2 、Weka 3.8.5

六、实验步骤：

1. 新建 main.py;
2. 定义 Apriori 类，包括初始化和函数操作;

```
class Apriori:
    # transition set
    traDatas = []
    # transition set's length
    traLen = 0
    # frequent k set, start with 1
```

实验二

```
k = 1
# counting the number of transition set
traCount = {}
# store frequent transition
freTran = {}
# support
sup = 0
# confidence
conf = 0
freAllTran = {}

def __init__(self, traDatas, sup, conf):
    self.traDatas = traDatas
    self.traLen = len(traDatas)
    self.sup = sup
    self.conf = conf

def scanFirDatas(self): # count frequency for each element
    tmpList = ','.join(traDatas).split(',')
    tmpSetList = [(each,) for each in tmpList]
    self.traCount = dict(collections.Counter(tmpSetList))
    return self.traCount

def getFreSet(self): # find event with higher support. and get frequent k set
    self.freTran = {}
    for tra in self.traCount.keys():
        if self.traCount[tra] >= self.sup and len(tra) == self.k:
            self.freTran[tra] = self.traCount[tra] # store frequent set
            self.freAllTran[tra] = self.traCount[tra]

# compare if k-l elements is equal
def cmpTwoSet(self, setA, setB) :
    setA = set(setA)
    setB = set(setB)
    if len(setA-setB) == 1 and len(setB-setA) == 1:
        return True
    else:
        return False

def selfConn(self): # connecting events. Only an element is added.
    self.traCount = {}
    for item in itertools.combinations(self.freTran.keys(),2):# connecting event
        between any two event.
        if self.cmpTwoSet(item[0], item[1]) == True: # only an element is added.
            key = item[0] + item[1]
```

实验二

```
        if self.cutBranch(key) != False:
            self.traCount[key] = 0

def cutBranch(self, key): # if subKey of the event is not frequent, return false
    for subKey in list(itertools.combinations(key, self.k)):
        if subKey not in self.freTran.keys():
            return False

def scanDatas(self):# count support
    self.k = self.k + 1
    for key in self.traCount.keys():
        for tra in traDatas:
            if set(key).issubset(tuple(tra.split(','))):
                self.traCount[key] += 1

def permutation2(self, string, pre_str, container):
    if len(string) == 1:
        container.append(pre_str + string)
    for idx, str in enumerate(string):
        new_str = string[:idx] + string[idx+1:]
        new_pre_str = pre_str + str
        self.permutation(new_str, new_pre_str, container)

def permutation(self, tup, pre_tup, container):
    if len(tup) == 1:
        container.append(pre_tup + tup)
    for idx, elem in enumerate(tup):
        new_tup = tup[:idx] + tup[idx+1:]
        new_pre_tup = pre_tup + (elem,)
        self.permutation(new_tup, new_pre_tup, container)

def genAssRule(self):
    container = []
    ruleSet = set()
    for item in self.freTran.keys():
        self.permutation(item, (), container)
    for item in container:
        for i in range(1, len(item)):
            ruleSet.add((item[:i], item[i:]))
    for rule in ruleSet:
        if self.calcConfi(rule[0], rule[1]) > self.conf:
            print (rule[0], end="_-->>>")
            print(rule[1])

def calcConfi(self, first, last): #computing confidence
```

实验二

```
if first+last not in self.freAllTran.keys():
    return self.freAllTran[last+first]/self.freAllTran[first]
return self.freAllTran[first + last] / self.freAllTran[first]

def algorithm (self):
    self.scanFirDats()# count frequency for each element
    while self.traCount != {}:
        self.getFreSet() # find event with higher support. and get frequent k set
        self.selfConn() # connecting events. Only an element is added.cut branch
        self.scanDats() # count support
    print(self.freAllTran)
    print(self.freTran)
    self.genAssRule()# mining rules.
```

3. 主函数，读取数据集并进行关联规则挖掘，k 设置为 2，置信度设置为 0.7。

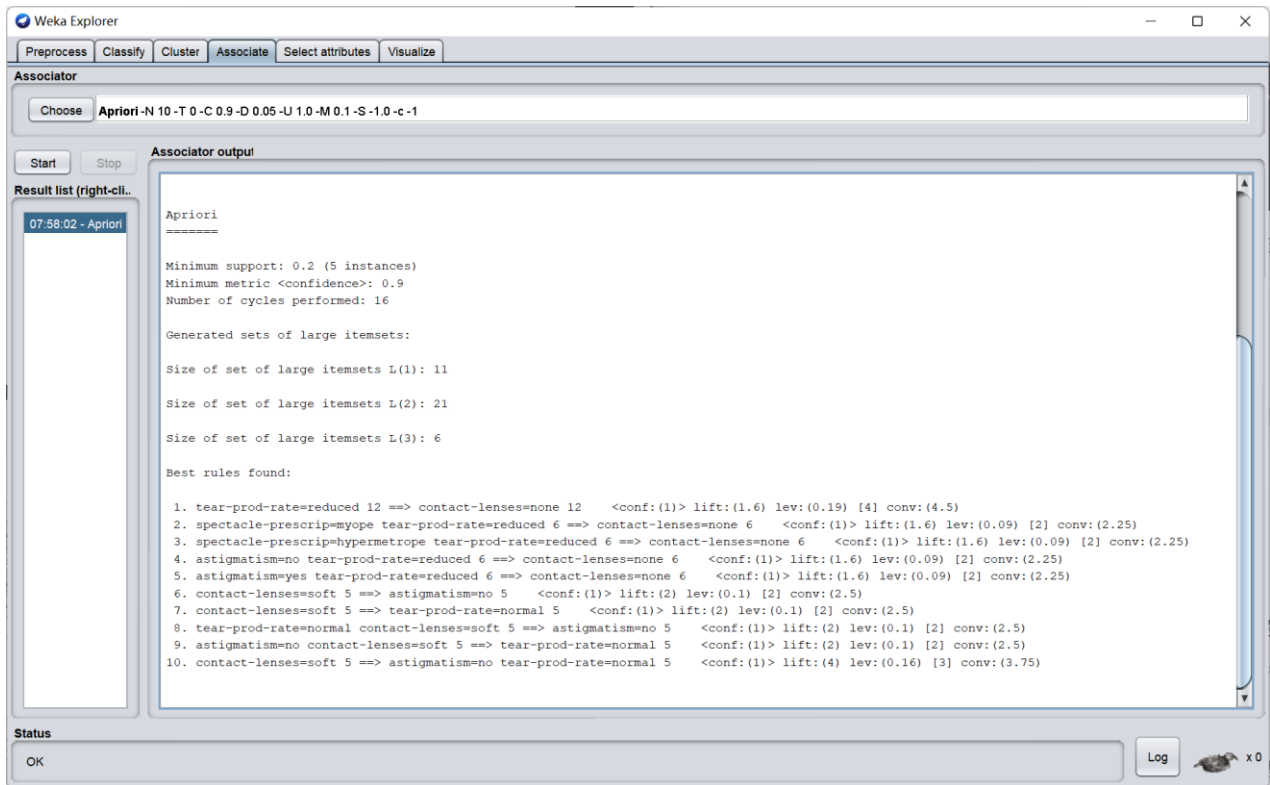
```
with open('contact-lenses.arff') as f:
    traDats = []
    perline = f.readline()[:-1]
    while perline:
        traDats.append(perline)
        perline = f.readline()[:-1]
    # print(traDats)
    apriori = Apriori(traDats, 2, 0.7)
    apriori.algorithm()
```

4. 运行代码，得到实验结果。

七、实验数据及结果分析：

使用 weka 图形界面实现关联规则挖掘，结果如下：

实验二



实验结果如图所示：

```
D:\Miniconda3\python.exe D:/Desktop/数据挖掘实验/DM-Experiments/DM-Exp2/main.py
{('young',): 8, ('myope',): 12, ('no',): 12, ('reduced',): 12, ('none',): 15, ('normal',): 12, ('soft',): 5, ('yes',): 12, ('hard',): 4, ('hyp
{('young', 'myope'): 4, ('young', 'no'): 4, ('young', 'reduced'): 4, ('young', 'none'): 4, ('young', 'normal'): 4, ('young', 'soft'): 2, ('you
('soft',)_-->>>('no',)
('reduced',)_-->>>('none',)
('hard',)_-->>>('myope',)
('hard',)_-->>>('yes',)
('presbyopic',)_-->>>('none',)
('soft',)_-->>>('normal',)
('hard',)_-->>>('normal',)
('none',)_-->>>('reduced',)
```

可见，算法成功地将数据集中各个元素出现的频次统计出来，并进行了关联规则挖掘，成功找到 8 组关联规则。

八、实验结论、心得体会和改进建议：

通过动手实现 Apriori 算法，对关联规则挖掘的一般方法进行了初步掌握，对关联规则挖掘算法有了更深入的理解，进一步强化了理论知识的学习，加深了自己对 Apriori 算法的印象。

在本次实验中，遇到的最大困难是选取合适的数据结构作为元素出现频次字典的键，由于一开始选取了元组而非集合，走了很多弯路，最终在列表、元组

实验二

与集合三种数据结构之间反复切换，总算达到了实验目标，最终顺利完成实验。感觉在解决这个问题过程中收获很大，除了对算法相应知识有了更进一步的掌握，还强化了自己在 `python` 编程方面的技能。

电子科技大学

实验报告

实验三

一、实验名称：分类（KNN）

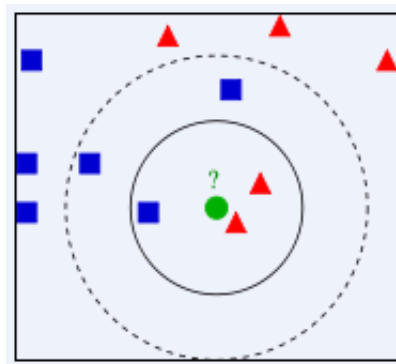
二、实验学时：4

三、实验内容和目的：

实现 KNN 算法

四、实验原理：

KNN 属于 lazy learning，不会对训练样本数据进行学习，其做法是：对于一个新数据，计算它与训练集中数据的距离，选择最短的 k 个作为邻居，然后预测它的类别和 k 个邻居中其所属类别最多的一致。



五、实验器材（设备、元器件）

笔记本 1 台，系统 Windows 11

Python 3.9、Pycharm 2021.2 、 Weka 3.8.5

六、实验步骤:

1. 新建 classifier.py;
2. 读取训练集, 如下:

```
import math
import re

dataset = []
dataLen = []
testDataSet = []

with open("iris.2D.train.arff") as fin:
    for line in fin.readlines():
        data = re.split(",", line.strip())
        dataLen = len(data)
        dataset.append(data)
```

3. 定义用于分类的函数, vote 函数用于投票得出标签, findNN 函数用于找到最邻近的 k 个数据, computeDis 函数用于计算几何距离。

```
def computeDis(x, y):
    return math.sqrt(math.pow(float(x[0])-float(y[0]), 2) +
math.pow(float(x[1])-float(y[1]), 2))

def findNN(testData, k):
    distances = []
    for data in dataset:
        distances.append(computeDis(testData, data))
    return (sorted(range(len(distances)), key=lambda m:distances[m]))[:k]

def vote(indexs):
    votes = {}
    for i in indexs:
        label = dataset[i][dataLen-1]
        if label not in votes.keys():
            votes[label] = 1
        else:
            votes[label] = votes[label]+1
```

实验三

```
return (max(votes, key=votes.get))
```

4. 读入测试集，通过 KNN 进行预测，并输出准确率。

```
if __name__ == "__main__":
    with open("iris.2D.test.arff") as ftest:
        for line in ftest.readlines():
            testData = re.split(",", line.strip())
            testData.append.vote(findNN([testData[0], testData[1]], 2))
            testDataSet.append(testData)

    total = 0
    right = 0
    for each in testDataSet:
        total += 1
        if each[2]==each[3]:
            right += 1

    print("%.2f"%(right/total*100),'%')
```

5. 运行该程序，观察运行结果。

七、实验数据及结果分析：

使用 weka 图形界面实现 KNN 分类，结果如下：

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The classifier chosen is 'IBK - K 1 - W 0 - A %weka.core.neighboursearch.LinearNNSearch - A %weka.core.EuclideanDistance - R first-last'. The test options are set to 'Cross-validation' with 'Folds' set to 10. The classifier output window displays the following results:

Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

| Metric | Value | Percentage |
|----------------------------------|-----------|------------|
| Correctly Classified Instances | 144 | 96 % |
| Incorrectly Classified Instances | 6 | 4 % |
| Kappa statistic | 0.94 | |
| Mean absolute error | 0.0279 | |
| Root mean squared error | 0.1322 | |
| Relative absolute error | 6.2681 % | |
| Root relative squared error | 28.0374 % | |
| Total Number of Instances | 150 | |

=== Detailed Accuracy By Class ===

| TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---------------|---------|-----------|--------|-----------|-------|----------|----------|-----------------|
| 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | Iris-setosa |
| 0.960 | 0.040 | 0.923 | 0.960 | 0.941 | 0.911 | 0.980 | 0.966 | Iris-versicolor |
| 0.920 | 0.020 | 0.958 | 0.920 | 0.939 | 0.910 | 0.982 | 0.938 | Iris-virginica |
| Weighted Avg. | 0.960 | 0.020 | 0.960 | 0.960 | 0.940 | 0.987 | 0.968 | |

=== Confusion Matrix ===

| a | b | c | <-- classified as |
|----|----|----|---------------------|
| 50 | 0 | 0 | a = Iris-setosa |
| 0 | 48 | 2 | b = Iris-versicolor |
| 0 | 4 | 46 | c = Iris-virginica |

通过改变 k 值进行多次实验，观察实验结果得到如下数据：

| k | Accuracy | k | Accuracy | k | Accuracy |
|---|----------|----|----------|----|----------|
| 1 | 97.33% | 6 | 96.00% | 11 | 97.33% |
| 2 | 97.33% | 7 | 96.00% | 12 | 97.33% |
| 3 | 96.00% | 8 | 97.33% | 13 | 98.67 % |
| 4 | 96.00% | 9 | 97.33% | 14 | 97.33% |
| 5 | 96.00% | 10 | 97.33% | 15 | 97.33% |

当 $k=1$ 或 2 时，预测准确率高达 97.33%，说明分类器工作得很好；当 $k \geq 3$ 时，准确率有所降低，变为 96%，而当 k 增加到 8 时，准确率重新回到 97.33%， $k=13$ 时，准确率达到极大值 98.67%。可见，KNN 分类的准确率跟 k 值的选取有关，但是无明显规律。

八、实验结论、心得体会和改进建议：

通过手动实现 KNN 算法，对其原理及思路有了更深刻的理解，对其细节有了更透彻的掌握，进一步强化了理论知识的学习，加深了自己对分类算法的印象。

这个实验确实挺简单的，建议以后不用给示例代码了。

电子科技大学

实 验 报 告

实验四

一、实验名称：聚类实验

二、实验学时：4

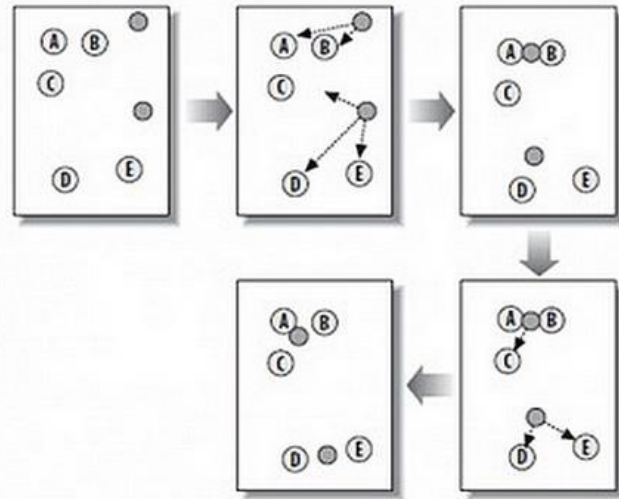
三、实验内容和目的：

1. 了解聚类的基本概念、原理和一般方法
2. 掌握聚类的基本算法
3. 学会调用 WEKA 包处理 kmeans 聚类问题；
4. 自己编程实现 K-Means、DBSCAN 算法；

四、实验原理：

1. K-Means 算法原理

- (1) 随机在图中取 K （如图 $K=2$ ）个聚类中心。
- (2) 然后对图中的所有点求到这 K 个种子点的距离，假如点 P_i 离聚类中心 S_i 最近，那么 P_i 属于 S_i 点群。接下来，我们要移动聚类中心到属于他的“点群”的中心。
- (3) 然后重复第(2)和第(3)步，直到聚类中心几乎不发生变化。



2. DBSCAN 算法原理

通过检查数据集中每个对象的 ϵ -邻域来寻找聚类。如果一个点 p 的 ϵ -邻域包含多于 MinPts 个对象，则创建一个 p 作为核心对象的新簇。然后，DBSCAN 反复地寻找从这些核心对象直接密度可达的对象，这个过程可能涉及一些密度可达簇的合并。当没有新的点可以被添加到任何簇时，该过程结束。具体如下：

输入：包含 n 个对象的数据库，半径 ϵ ，最少数目 MinPts 。

输出：所有生成的簇，达到密度要求。

a. REPEAT

b. 从数据库中抽取一个未处理过的点；

c. IF 抽出的点是核心点 THEN 找出所有从该点密度可达的对象，形成一个簇

d. ELSE 抽出的点是边缘点(非核心对象)，跳出本次循环，寻找下一点；

e. UNTIL 所有点都被处理；

五、实验器材（设备、元器件）

笔记本 1 台，系统 Windows 11

Python 3.9、Pycharm 2021.2 、Weka 3.8.5

六、实验步骤：

1. 新建 kmeans.py;
2. 加载数据集操作:

```
def loadDataset(infile):
    f = open(infile, "r")
    line = f.readline()
    line = line[:-1]
    lines = []
    lines2 = []
    while line: # 直到读取完文件
        line = f.readline() # 读取一行文件，包括换行符
        line = line[:-1] # 去掉换行符，也可以不去
        lines.append(line)
    f.close() # 关闭文件
    lines = lines[:-1]
    for x in lines:
        a,b = x.split(" ")
        a = float(a)
        b = float(b)
        lines2.append([a,b])
    t = np.array(lines2)
    return t
```

3. 定义 Kmeanscluster 类，及相应的函数:

```
class KMeansCluster():
    def __init__(self, k=3, initCent = 'random', max_iter = 500):
        self._k = k
        self._initCent = initCent
        self._max_iter = max_iter
        self._clusterAssment = None
        self._labels = None

    def _calEDist(self, arrA, arrB):
        return np.math.sqrt(sum(np.power(arrA - arrB, 2)))

    def _randCent(self, data_X, k):
        n = data_X.shape[1] # 获取特征的维数
        centroids = np.empty((k, n)) # 使用 numpy 生成- 一个 k*n 的矩阵，用于存储质心
        for j in range(n):
            minJ= min(data_X[:,j])
```

实验四

```
        rangeJ = float(max(data_X[:, j]-minJ))
# 使用flatten 拉平嵌套列表(nested List)
        centroids[:, j] = (minJ + rangeJ * np.random.rand(k,1)).flatten()
    return centroids

def fit(self, data_X):
    m = data_X.shape[0] # 获取样本的个数
    # 一个m*2的维矩阵，矩阵第- 列存储样本点所属的族的索引值，
    # 第二列存储该点与所属族的质心的平方误差
    self._clusterAssment = np.zeros((m, 2))
    if self._initCent == "random" :
        self._centroids = self._randCent(data_X, self._k)

    clusterChanged = True
    for _ in range(self._max_iter):
        clusterChanged = False
        for i in range(m): # 将每个样本点分配到离它最近的质心所属的族
            minDist = np.inf # 首先将minDist 置为一个无穷大的数
            minIndex = -1 # 将最近质心的下标置为-1
            for j in range(self._k): # 次迭代用于寻找最近的质心
                arrA = self._centroids[j, :]
                arrB = data_X[i, :]
                distJI = self._calEDist(arrA, arrB) # 计算误差值
                if distJI < minDist:
                    minDist = distJI
                    minIndex = j
            if self._clusterAssment[i, 0] != minIndex or self._clusterAssment[i,
1] > minDist ** 2:
                clusterChanged = True
                self._clusterAssment[i, :] = minIndex, minDist**2
        if not clusterChanged: # 若所有样本点所属的族都不改变,则已收敛,结束迭代
            break
        for i in range(self._k): # 更新质心, 将每个族中的点的均值作为质心
            index_all = self._clusterAssment[:,0] # 取出样本所属簇的索引值
            value = np.nonzero(index_all == i) # 取出所有属于第i个簇的索引值
            ptsInClust = data_X[value[0]] # 取出属于第1个族的所有样本点
            self._centroids[i, :] = np.mean(ptsInClust, axis = 0)
    self._labels = self._clusterAssment[:,0]

    return self._centroids, self._labels
```

4. 主函数:

```
if __name__ == "__main__":
```

```

data_X = loadDataset("data.txt")
k = 3
clf = KMeansCluster(k)
cents, labels = clf.fit(data_X)
colors = ['b', 'g', 'r', 'k', 'C', 'm', 'y', '#e24fff', '#524C90', '#845868']
for i in range(k):
    index = np.nonzero(labels == i)[0]
    x0 = data_X[index, 0]
    x1 = data_X[index, 1]
    y_i = i
    for j in range(len(x0)):
        plt.text(x0[j], x1[j], str(y_i), color = colors[i], fontdict = {'weight':
'bold', 'size': 6})
    plt.scatter(cents[i, 0], cents[i, 1], marker = 'x', color = colors[i], linewidths
= 7)
plt.axis([-50, 150, -50, 150])
plt.show()

```

5. 新建 dbscan.py;

6. 定义加载数据集操作;

```

def loadDataSet(fileName, splitChar='\t'):
    dataSet = []
    with open(fileName) as fr:
        for line in fr.readlines():
            curline = line.strip().split(splitChar)
            fltline = list(map(float, curline))
            dataSet.append(fltline)
    return dataSet

```

7. 定义求欧氏距离、判断是否为邻近点等基本操作;

```

def dist(a, b):
    return math.sqrt(np.power(a-b, 2).sum())

def eps_neighbor(a, b, eps):
    return dist(a, b) < eps

def region_query(data, pointId, eps):
    nPoints = data.shape[1]
    seeds = []
    for i in range(nPoints):
        if eps_neighbor(data[:, pointId], data[:, i], eps):
            seeds.append(i)

```

```
return seeds
```

8. 定义 expand_cluster 函数;

```
def expand_cluster(data, clusterResult, pointId, clusterId, eps, minPts):
    seeds = region_query(data, pointId, eps)
    if len(seeds) < minPts:
        clusterResult[pointId] = NOISE
        return False
    else:
        clusterResult[pointId] = clusterId
        for seedId in seeds:
            clusterResult[seedId] = clusterId

        while len(seeds) > 0:
            currentPoint = seeds[0]
            queryResults = region_query(data, currentPoint, eps)
            if len(queryResults) >= minPts:
                for i in range(len(queryResults)):
                    resultPoint = queryResults[i]
                    if clusterResult[resultPoint] == UNCLASSIFIED:
                        seeds.append(resultPoint)
                        clusterResult[resultPoint] = clusterId
                    elif clusterResult[resultPoint] == NOISE:
                        clusterResult[resultPoint] = clusterId
            seeds = seeds[1:]
        return True
```

9. 定义 dbscan 函数;

```
def dbscan(data, eps, minPts):
    clusterId = 1
    nPoints = data.shape[1]
    clusterResult = [UNCLASSIFIED] * nPoints
    for pointId in range(nPoints):
        point = data[:,pointId]
        if clusterResult[pointId] == UNCLASSIFIED:
            if expand_cluster(data, clusterResult, pointId, clusterId, eps, minPts):
                clusterId = clusterId + 1
    return clusterResult, clusterId - 1
```

10. 定义函数用于绘制散点图;

```
def plotFeature(data, clusters ,clusterNum):
    nPoints = data.shape[1]
    matClusters = np.mat(clusters).transpose()
```

实验四

```
fig = plt.figure()
scatterColors = ['black', 'blue', 'green', 'yellow', 'red', 'purple', 'orange',
'brown']
ax = fig.add_subplot(111)
for i in range(clusterNum + 1):
    colorStyle = scatterColors[i % len(scatterColors)]
    subCluster = data[:,np.nonzero(matClusters[:,0].A == i)]
    ax.scatter(subCluster[0,:].flatten().A[0], subCluster[1,:].flatten().A[0],
c=colorStyle, s=50)
```

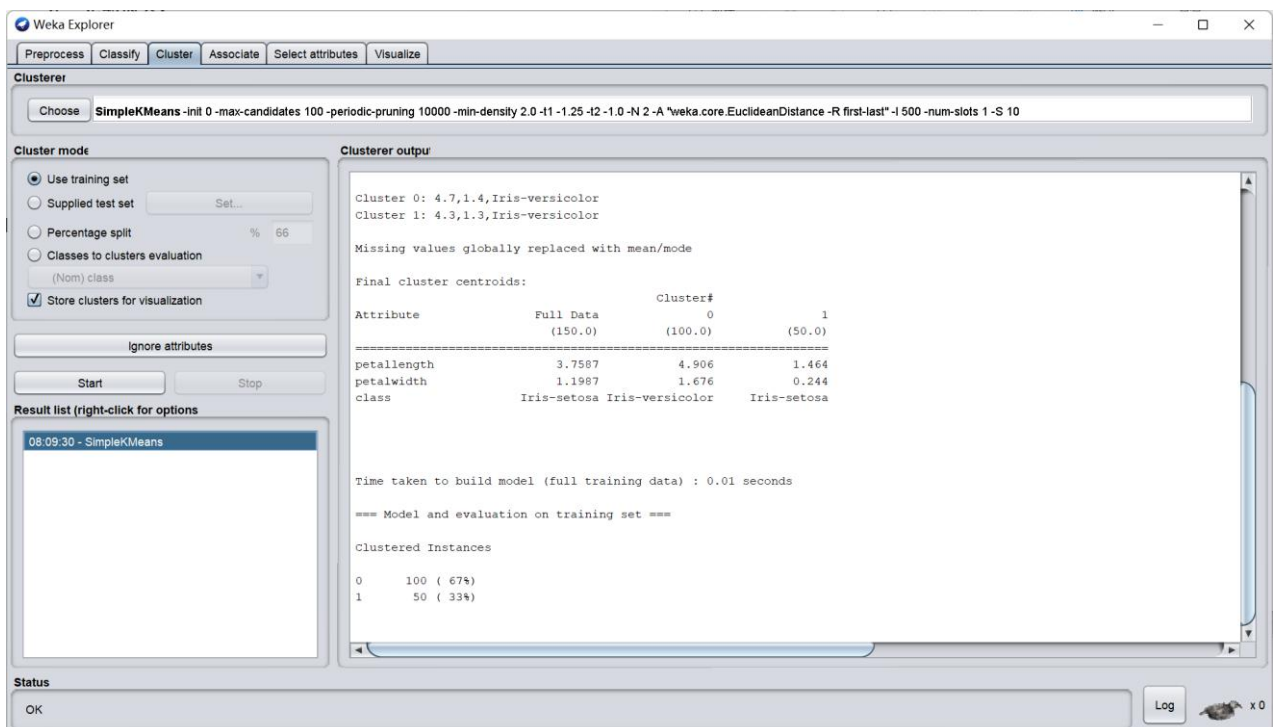
11. 主函数。

```
def main():
    dataSet = loadDataSet('data.txt', splitChar=' ')
    dataSet = np.mat(dataSet).transpose()
    clusters, clusterNum = dbscan(dataSet, 2, 15)
    print("cluster Numbers = ", clusterNum)
    print (clusters)
    plotFeature(dataSet, clusters, clusterNum)
```

12. 运行该程序，观察运行结果。

七、实验数据及结果分析：

使用 weka 图形界面实现关联规则挖掘，结果如下：



The screenshot shows the Weka Explorer application window. The 'Clusterer' tab is selected. The 'Choose' dropdown is set to 'SimpleKMeans-init 0-max-candidates 100-periodic-pruning 10000-min-density 2.0-t1-1.25-t2-1.0-N 2-A "weka.core.EuclideanDistance"-R first-last-I 500-num-slots 1-S 10'. The 'Cluster mode' section has 'Use training set' selected. The 'Clusterer output' pane displays the following results:

Cluster 0: 4.7,1.4,Iris-versicolor
Cluster 1: 4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode

Final cluster centroids:

| Attribute | Full Data (150.0) | Cluster# 0 (100.0) | 1 (50.0) |
|-------------|-----------------------------|-----------------------|-------------|
| petallength | 3.7587 | 4.906 | 1.464 |
| petalwidth | 1.1987 | 1.676 | 0.244 |
| class | Iris-setosa Iris-versicolor | Iris-setosa | |

Time taken to build model (full training data) : 0.01 seconds

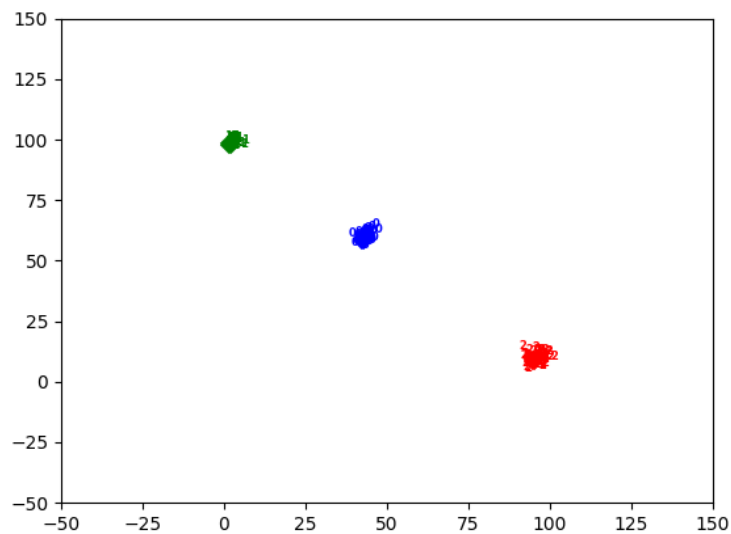
=== Model and evaluation on training set ===

Clustered Instances

| Cluster | Count | Percentage |
|---------|-------|------------|
| 0 | 100 | (67%) |
| 1 | 50 | (33%) |

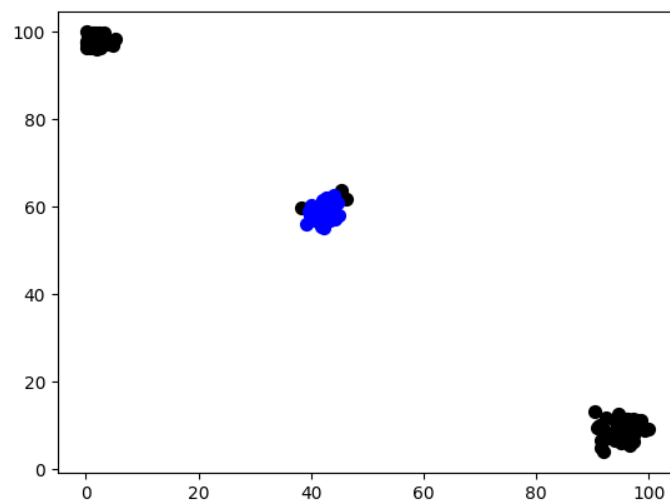
The 'Result list (right-click for options)' pane shows '08:09:30 - SimpleKMeans'. The 'Status' bar at the bottom indicates 'OK'.

通过运行 `kmeans.py`，得到如下结果：



可见，通过 `kmeans` 分类，成功将数据集分成三类，结果符合预期。

通过运行 `dbscan.py`，得到如下结果：



八、实验结论、心得体会和改进建议：

通过手动实现 `kmeans` 算法，对 `kmeans` 分类有了更深刻的理解，对其细节有了更透彻的掌握，进一步强化了理论知识的学习，加深了自己对分类算法的

印象。

无改进建议。