

电子科技大学
计算机科学与工程学院

标准实验报告

(实验) 课程名称 数据挖掘与大数据分析

电子科技大学教务处制表

电子科技大学

电子科技大学

实验报告

学生姓名：蔡与望 学号：2020010801024 指导教师：蔡世民

实验地点：主楼 A2-413A

实验时间：2022/11/15

一、实验室名称：计算机学院实验中心

二、实验项目名称：认识数据与数据预处理

三、实验学时：2 学时

四、实验原理：

现实世界的数据是不完整的、含噪声的、不一致的。没有高质量的数据，就没有高质量的挖掘结果。为了保证数据的准确性、完整性、一致性、时效性、可信性、可解释性，我们需要进行数据预处理。

归一化是数据变换的一种。它将量纲不同、标准不同的数据，规范化进一个新的区间，便于相互比较。其常用公式为：

$$v' = \frac{v - \min A}{\max A - \min A} (\text{newmax}A - \text{newmin}A) + \text{newmin}A$$

这将原本在 $(\min A, \max A)$ 区间内的数据，归一化到了 $(\text{newmin}A, \text{newmax}A)$ 区间内。

缺失值处理是数据清理的一种。它将不完整的数据集中的空缺值，通过推断而填充补全。有两种常见的填充策略，第一种是用属性平均值填充，即将缺失值所属属性的现有数据的平均值作为缺失值；第二种是用最可能的值填充，即找出与缺失值的数据最相似的数据，用这些数据的平均值作为缺失值。

特征筛选是数据归约的一种。它通过删除不相干的属性，减少要处理的数据量。最常用的方法是信息增益和 ID3。它们通过计算总数据集的信息熵和各特征的信息熵，筛选出信息增益最大的特征属性，并将它们作为主要属性来处理。

五、实验目的：

回顾数据预处理的常见角度和方法，熟悉并掌握归一化、缺失值处理和特征

筛选的原理以及代码实现。

六、实验内容：

1. 归一化。
2. 缺失值处理。
3. 特征筛选。
4. 数据可视化。

七、实验器材（设备、元器件）：

PC 微机一台。

八、实验步骤：

1. 归一化

对所有实例使用归一化过滤器。

```
private void normalize() throws Exception {
    Normalize norm = new Normalize();
    norm.setInputFormat.instances);
    instances = Filter.useFilter(instances, norm);
}
```

2. 缺失值处理

获取每一属性的平均值（numeric）或众数（nominal）。

```
private double[] getSubstitutions() {
    int columnNum = instances.numAttributes();
    double[] substitutions = new double[columnNum];

    for (int column = 0; column < columnNum; column++) {
        substitutions[column] = instances.meanOrMode(column);
    }

    return substitutions;
}
```

用替代值替代缺失值。

```
private void substitute() {
    double[] substitutions = getSubstitutions();

    int columnNum = instances.numAttributes();
    for (Instance instance : instances) {
        for (int column = 0; column < columnNum; column++) {
            instance.replaceMissingValues(substitutions);
        }
    }
}
```

3. 特征筛选

采用信息增益评估，挑选前若干个属性。

```
private void select(int featureNum) throws Exception {
    AttributeSelection selection = new AttributeSelection();
    selection.setInputFormat(instances);

    ASEvaluation evaluation = new InfoGainAttributeEval();
    selection.setEvaluator(evaluation);

    Ranker rank = new Ranker();
    rank.setThreshold(0);
    rank.setNumToSelect(featureNum);
    selection.setSearch(rank);

    instances = Filter.useFilter(instances, selection);
}
```

4. 数据可视化

使用 seaborn 绘制散点图和直方图。

```
def scatter(group: str):
    sns.relplot(
        data=iris,
        x=f"{group}width",
        y=f"{group}length",
        hue="class",
        palette=("red", "green", "blue"),
    )
    plt.savefig(f"outputs/iris-{group}-scatter.png")
    plt.clf()
```

```
def hist(attribute: str):
    sns.histplot(
        data=iris,
        x=attribute,
        hue="class",
        multiple="stack",
    )
    plt.savefig(f"outputs/iris-{attribute}-histogram.png")
    plt.clf()
```

九、实验数据及结果分析：

1. 归一化

使用 weka 对 iris.arff 进行归一化处理，结果如下。

Viewer

Relation: iris-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0

No.	1: sepalength Numeric	2: sepalwidth Numeric	3: petallength Numeric	4: petalwidth Numeric	5: class Nominal
1	0.222222222222222213	0.62499999999999999	0.06779661016949151	0.041666666666666667	Iris-setosa
2	0.16666666666666668	0.41666666666666663	0.06779661016949151	0.041666666666666667	Iris-setosa
3	0.11111111111111119	0.5	0.05084745762711865	0.041666666666666667	Iris-setosa
4	0.083333333333333327	0.45833333333333333	0.0847457627118644	0.041666666666666667	Iris-setosa
5	0.19444444444444448	0.66666666666666666	0.06779661016949151	0.041666666666666667	Iris-setosa
6	0.30555555555555564	0.79166666666666665	0.11864406779661016	0.12500000000000003	Iris-setosa
7	0.083333333333333327	0.58333333333333333	0.06779661016949151	0.08333333333333333	Iris-setosa
8	0.19444444444444448	0.58333333333333333	0.0847457627118644	0.041666666666666667	Iris-setosa
9	0.027777777777777922	0.37499999999999999	0.06779661016949151	0.041666666666666667	Iris-setosa
10	0.16666666666666668	0.45833333333333333	0.0847457627118644	0.0	Iris-setosa
11	0.30555555555555564	0.70833333333333333	0.0847457627118644	0.041666666666666667	Iris-setosa
12	0.13888888888888887	0.58333333333333333	0.1016949152542373	0.041666666666666667	Iris-setosa
13	0.13888888888888887	0.41666666666666663	0.06779661016949151	0.0	Iris-setosa
14	0.0	0.41666666666666663	0.016949152542372895	0.0	Iris-setosa
15	0.41666666666666663	0.83333333333333333	0.033898305084745756	0.041666666666666667	Iris-setosa
16	0.38888888888888895	1.0	0.0847457627118644	0.12500000000000003	Iris-setosa
17	0.30555555555555564	0.79166666666666665	0.05084745762711865	0.12500000000000003	Iris-setosa
18	0.222222222222222213	0.62499999999999999	0.06779661016949151	0.08333333333333333	Iris-setosa
19	0.38888888888888895	0.74999999999999998	0.11864406779661016	0.08333333333333333	Iris-setosa
20	0.222222222222222213	0.74999999999999998	0.0847457627118644	0.08333333333333333	Iris-setosa
21	0.30555555555555564	0.58333333333333333	0.11864406779661016	0.041666666666666667	Iris-setosa
22	0.222222222222222213	0.70833333333333333	0.0847457627118644	0.12500000000000003	Iris-setosa
23	0.083333333333333327	0.66666666666666666	0.0	0.041666666666666667	Iris-setosa
24	0.222222222222222213	0.54166666666666665	0.11864406779661016	0.16666666666666669	Iris-setosa
25	0.13888888888888887	0.58333333333333333	0.15254237288135591	0.041666666666666667	Iris-setosa
26	0.19444444444444448	0.41666666666666663	0.1016949152542373	0.041666666666666667	Iris-setosa
27	0.19444444444444448	0.58333333333333333	0.1016949152542373	0.12500000000000003	Iris-setosa
28	0.25000000000000006	0.62499999999999999	0.0847457627118644	0.041666666666666667	Iris-setosa
29	0.25000000000000006	0.58333333333333333	0.06779661016949151	0.041666666666666667	Iris-setosa
30	0.11111111111111119	0.5	0.1016949152542373	0.041666666666666667	Iris-setosa

Add instance Undo OK Cancel

使用 Java 对 iris.arff 进行归一化处理，结果如下。

```
iris-normalized.arff X
outputs > preprocessing > iris-normalized.arff
9 @data
10 0.222222,0.625,0.067797,0.041667,Iris-setosa
11 0.166667,0.416667,0.067797,0.041667,Iris-setosa
12 0.111111,0.5,0.050847,0.041667,Iris-setosa
13 0.083333,0.458333,0.084746,0.041667,Iris-setosa
14 0.194444,0.666667,0.067797,0.041667,Iris-setosa
15 0.305556,0.791667,0.118644,0.125,Iris-setosa
16 0.083333,0.583333,0.067797,0.083333,Iris-setosa
17 0.194444,0.583333,0.084746,0.041667,Iris-setosa
18 0.027778,0.375,0.067797,0.041667,Iris-setosa
19 0.166667,0.458333,0.084746,0,Iris-setosa
20 0.305556,0.708333,0.084746,0.041667,Iris-setosa
21 0.138889,0.583333,0.101695,0.041667,Iris-setosa
22 0.138889,0.416667,0.067797,0,Iris-setosa
23 0,0.416667,0.016949,0,Iris-setosa
24 0.416667,0.833333,0.033898,0.041667,Iris-setosa
25 0.388889,1,0.084746,0.125,Iris-setosa
26 0.305556,0.791667,0.050847,0.125,Iris-setosa
27 0.222222,0.625,0.067797,0.083333,Iris-setosa
28 0.388889,0.75,0.118644,0.083333,Iris-setosa
29 0.222222,0.75,0.084746,0.083333,Iris-setosa
30 0.305556,0.583333,0.118644,0.041667,Iris-setosa
31 0.222222,0.708333,0.084746,0.125,Iris-setosa
32 0.083333,0.666667,0,0.041667,Iris-setosa
33 0.222222,0.541667,0.118644,0.166667,Iris-setosa
34 0.138889,0.583333,0.152542,0.041667,Iris-setosa
35 0.194444,0.416667,0.101695,0.041667,Iris-setosa
36 0.194444,0.583333,0.101695,0.125,Iris-setosa
```

可以看到，两者结果一致。

2. 缺失值处理

使用 weka 查看 labor.arff 原数据，结果如下。

Relation: labor-neg-data											
1: duration	2: wage-increase-first-year	3: wage-increase-second-year	4: wage-increase-third-year	5: cost-of-living-adjustment	6: working-hours	7: pension-status	8: stand-by-pay	9: shift-differential	10: education-allowance	11: statutor-holidays	
1 1.0	5.0				40.0			2.0		11.0	
2 2.0	4.5		5.0		35.0	ret_allw			res	11.0	
3					38.0	empl_contr			5.0		11.0
4 3.0	3.7		4.0	5.0tc					res		
5 3.0	4.5		4.5	5.0	40.0						12.0
6 2.0	2.0	2.5			35.0				6.0yes		12.0
7 3.0	4.0		5.0	5.0tc		empl_contr					12.0
8 3.0	6.9		4.8	2.3	40.0				3.0		12.0
9 2.0	5.0		7.0		38.0		12.0		25.0res		11.0
10 1.0	5.7			none	40.0	empl_contr			4.0		11.0
11 3.0	3.5		4.0	4.0none	38.0						13.0
12 2.0	6.4		6.4		38.0				4.0		15.0
13 2.0	2.5	4.0		none	40.0				2.0no		10.0
14 3.0	3.5		4.0	5.1tcf	37.0				4.0		13.0
15 1.0	3.0			none	36.0				10.0no		11.0
16 2.0	4.5		4.0	none	37.0	empl_contr					11.0
17 1.0	2.0			none	35.0				2.0		12.0
18 1.0	2.1			tc	40.0	ret_allw	2.0		3.0no		9.0
19 1.0	2.0			none	38.0	none			res		11.0
20 2.0	4.0	5.0		tcf	35.0		13.0		5.0		13.0
21 2.0	4.3	4.4			38.0				4.0		12.0
22 2.0	2.5	3.0			40.0	none					11.0
23 3.0	3.5	4.0		4.0tcf	27.0						10.0
24 2.0	4.5	4.0			40.0				4.0		10.0
25 1.0	6.0				38.0		8.0		3.0		9.0
26 3.0	2.0	2.0	2.0	2.0none	40.0	none					10.0
27 2.0	4.5		4.5	tcf					res		10.0
28 2.0	3.0	3.0	3.0	none	33.0				res		12.0
29 2.0	5.0	4.0		none	37.0				5.0no		11.0
30 3.0	2.0	2.5			35.0	none					10.0
31 3.0	4.5	4.5		5.0none	40.0				no		11.0
32 3.0	3.0	2.0	2.5tc		40.0	none			5.0no		10.0
33 2.0	2.5	2.5			38.0	empl_contr					10.0
34 2.0	4.0	5.0		none	40.0	none			3.0no		10.0
35 3.0	2.0	2.5		2.1tc	40.0	none			1.0no		10.0
36 2.0	2.0	2.0		none	40.0	none	2.0		no		11.0
37 1.0	2.0			tc	40.0	ret_allw	4.0		0.0no		11.0
38 1.0	2.0			tc	38.0	empl_contr	2.0		3.0no		9.0
39 3.0	2.0	2.5	2.0	2.0	37.0	empl_contr					10.0
40 2.0	4.5	4.0		none	40.0				4.0		12.0
41 1.0	4.0			none		none			res		11.0
42 2.0	2.0	3.0		none	38.0	empl_contr			res		12.0
43 2.0	2.5	2.5		tc	39.0	empl_contr					12.0
44 2.0	2.5	3.0		tcf	40.0	none					11.0
45 2.0	4.0	4.0		none	40.0	none			3.0		10.0
46 2.0	4.5	4.0			40.0				5.0no		10.0

使用 Java 对 labor.arff 进行缺失值处理，结果在 weka 中如下。

View											
Relation: labor-peg-data											
1:	duration	2:	3:	4:	5:	6:	7: pension	8:	9:	10:	11:
Header	Header	Header	Header	Header	Header	Header	Header	Header	Header	Header	Header
1	1.0	3.0	3.971739	3.913333none	40.0 emp_contr	7.444444	2.0no	4.0no	11.0		
2	2.0	4.5	3.913333none	5.0	35.0 ret_allw	7.444444	4.0no	4.0no	11.0		
3	2.160714	3.803971	3.971739	3.913333none	38.039716 emp_contr	7.444444	5.0no	4.0no	11.0		
4	3.0	3.7	4.0	5.0tc	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
5	3.0	4.5	5.0	5.0none	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
6	2.0	3.0	2.5	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
7	3.0	4.0	5.0	5.0tc	38.039716 emp_contr	7.444444	4.0no	4.0no	11.0		
8	3.0	6.9	4.0	2.0none	40.0 emp_contr	7.444444	3.0no	4.0no	11.0		
9	2.0	3.0	7.0	3.913333none	35.0 emp_contr	7.444444	2.0no	4.0no	11.0		
10	1.0	5.7	3.971739	3.913333none	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
11	3.0	3.5	4.0	4.0none	35.0 emp_contr	7.444444	3.0no	4.0no	11.0		
12	2.0	6.4	4.0	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
13	2.0	3.5	4.0	3.913333none	40.0 emp_contr	7.444444	2.0no	4.0no	11.0		
14	3.0	3.5	4.0	5.1tc	37.0 emp_contr	7.444444	4.0no	4.0no	11.0		
15	1.0	3.0	3.971739	3.913333none	36.0 emp_contr	7.444444	10.0no	4.0no	11.0		
16	2.0	4.5	4.0	3.913333none	37.0 emp_contr	7.444444	4.0no	4.0no	11.0		
17	1.0	2.8	3.971739	3.913333none	35.0 emp_contr	7.444444	2.0no	4.0no	11.0		
18	1.0	3.1	3.971739	3.913333tc	40.0 ret_allw	2.0	3.0no	4.0no	9.0		
19	1.0	2.0	3.971739	3.913333none	35.0 none	7.444444	4.0no	4.0no	11.0		
20	2.0	4.0	5.0	3.913333tc	35.0 emp_contr	7.444444	13.0	4.0no	11.0		
21	2.0	4.3	4.4	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
22	2.0	2.5	3.0	3.913333none	40.0 none	7.444444	4.0no	4.0no	11.0		
23	2.0	3.5	4.0	4.0tc	37.0 emp_contr	7.444444	4.0no	4.0no	11.0		
24	2.0	4.5	4.0	3.913333none	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
25	1.0	6.0	3.971739	3.913333none	35.0 emp_contr	7.444444	8.0	4.0no	9.0		
26	3.0	2.0	2.0	2.0none	40.0 none	7.444444	4.0no	4.0no	11.0		
27	2.0	4.5	4.0	3.913333tc	38.039716 emp_contr	7.444444	4.0no	4.0no	11.0		
28	2.0	3.0	3.0	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
29	2.0	3.0	4.0	3.913333none	37.0 emp_contr	7.444444	5.0no	4.0no	11.0		
30	2.0	2.5	2.5	3.913333none	35.0 none	7.444444	4.0no	4.0no	11.0		
31	3.0	4.5	4.5	5.0none	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
32	3.0	3.0	2.0	2.5tc	40.0 none	7.444444	5.0no	4.0no	11.0		
33	2.0	2.5	2.5	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
34	2.0	5.0	5.0	3.913333none	40.0 none	7.444444	3.0no	4.0no	11.0		
35	3.0	2.0	2.5	2.1tc	40.0 none	2.0	1.0no	4.0no	11.0		
36	2.0	3.0	2.0	3.913333none	40.0 none	7.444444	4.0no	4.0no	11.0		
37	1.0	2.0	3.971739	3.913333tc	40.0 ret_allw	4.0	5.0no	4.0no	11.0		
38	1.0	4.0	3.971739	3.913333none	35.0 emp_contr	7.444444	2.0	4.0no	9.0		
39	3.0	3.0	2.5	2.0none	37.0 emp_contr	7.444444	4.0no	4.0no	11.0		
40	2.0	4.5	4.0	3.913333none	40.0 emp_contr	7.444444	4.0no	4.0no	11.0		
41	1.0	4.0	3.971739	3.913333none	38.039716 none	7.444444	4.0no	4.0no	11.0		
42	2.0	3.0	3.0	3.913333none	35.0 emp_contr	7.444444	4.0no	4.0no	11.0		
43	2.0	2.5	2.5	3.913333tc	39.0 emp_contr	7.444444	4.0no	4.0no	11.0		
44	2.0	3.5	3.0	3.913333tc	40.0 none	7.444444	4.0no	4.0no	11.0		
45	2.0	4.0	4.0	3.913333none	40.0 none	7.444444	2.0no	4.0no	11.0		
46	2.0	4.5	4.0	3.913333none	40.0 none	7.444444	2.0no	4.0no	11.0		

Add instance

Undo

OK

Cancel

可以看到，所有属性的缺失值都被填充了。

3. 特征筛选

使用 weka 对 iris.arff 进行特征筛选，结果如下。

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Attribute Evaluator

Choose InfoGainAttributeEval

Search Method

Choose Ranker -T 0.0 -N 2

Attribute Selection Mode

Use full training set

Cross-validation Folds 10 Seed 1

No class

Start Stop

Result list (right-click for options)

1.418 3 petalwidth

1.378 4 petalwidth

Attribute selection output

=== Run information ===

Evaluator: weka.attributeSelection.InfoGainAttributeEval

Search: weka.attributeSelection.Ranker -T 0.0 -N 2

Relation: iris

Instances: 150

Attributes: 5

Attributes: sepalwidth sepalwidth petalwidth petalwidth class

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method: Attribute ranking.

Threshold for discarding attributes: 0

Attribute Evaluator (supervised, Class (nominal): 5 class): Information Gain Ranking Filter

Ranked attributes: 1.418 3 petalwidth 1.378 4 petalwidth

Selected attributes: 3,4 : 2

Status

Log

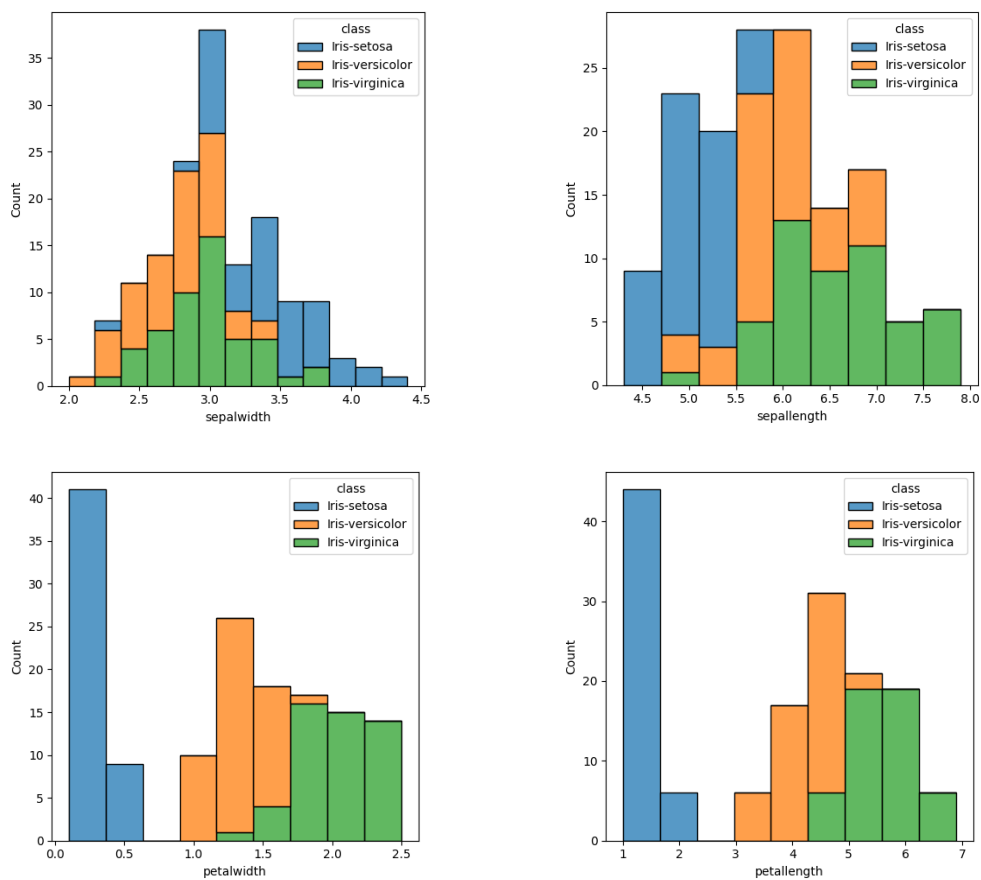
使用 Java 对 iris.arff 进行特征筛选，结果如下。

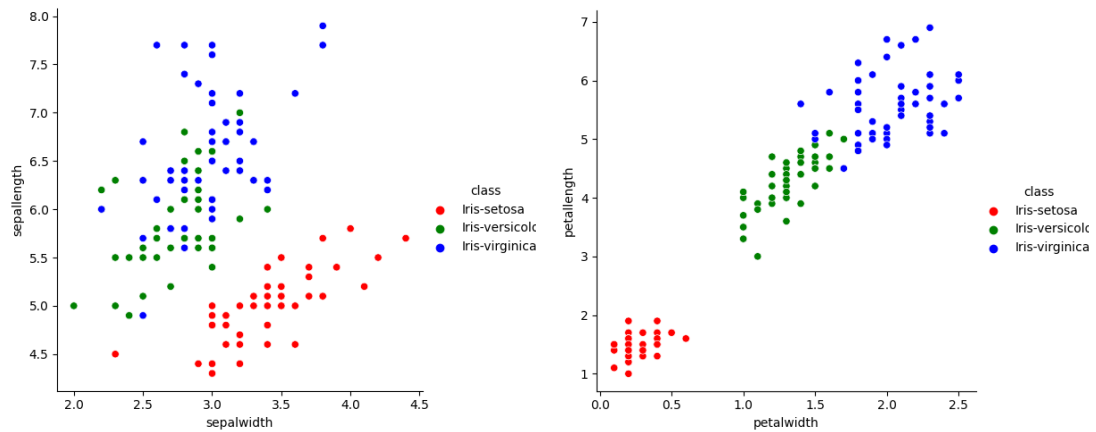
```
iris-selected.aff
outputs > preprocessing > iris-selected.aff
1 @relation 'iris-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.InfoGainAttributeEv2
2
3 @attribute petallength numeric
4 @attribute petalwidth numeric
5 @attribute class {Iris-setosa,Iris-versicolor,Iris-virginica}
6
7 @data
8 1.4,0.2,Iris-setosa
9 1.4,0.2,Iris-setosa
10 1.3,0.2,Iris-setosa
11 1.5,0.2,Iris-setosa
12 1.4,0.2,Iris-setosa
13 1.7,0.4,Iris-setosa
14 1.4,0.3,Iris-setosa
15 1.5,0.2,Iris-setosa
16 1.4,0.2,Iris-setosa
17 1.5,0.1,Iris-setosa
18 1.5,0.2,Iris-setosa
19 1.6,0.2,Iris-setosa
20 1.4,0.1,Iris-setosa
21 1.1,0.1,Iris-setosa
22 1.2,0.2,Iris-setosa
23 1.5,0.4,Iris-setosa
24 1.3,0.4,Iris-setosa
25 1.4,0.3,Iris-setosa
26 1.7,0.3,Iris-setosa
27 1.5,0.3,Iris-setosa
28 1.7,0.2,Iris-setosa
```

可以看到，两者最后都筛选出了 petallength、petalwidth 这两个特征属性。

4. 数据可视化

使用 Python 对 iris.csv 进行数据可视化，结果如下。





十、实验结论：

1. 归一化的编程结果与 GUI 结果一致。
2. 缺失值处理能够正确填充缺失值。
3. 特征筛选筛选出了 petallength、petalwidth 两个属性。
4. 数据可视化用直方图和散点图成功绘制了各花的花瓣、花萼长宽信息。

十一、总结及心得体会：

通过本实验，我对数据预处理的方法有了全面的回顾，熟悉并掌握了其中归一化、缺失值处理、特征筛选和数据可视化四方面的原理和实践。同时，我也学会了 Java 中 weka 包的调用，以及如何使用编程方式快速地处理数据。

十二、对本实验过程及方法、手段的改进建议：

无。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：蔡与望 学号：2020010801024 指导教师：蔡世民

实验地点：主楼 A2-413A

实验时间：2022/11/22

一、实验室名称：计算机学院实验中心

二、实验项目名称：关联规则挖掘

三、实验学时：2 学时

四、实验原理：

为了在事务、关系型数据库中的项集和对象中发现频繁模式、关联规则、相关性或因果结构，我们需要进行关联规则挖掘，即找到支持度和置信度足够大的所有规则。

支持度，指包含项集的事务数占总事务的比值。置信度，指在前件成立时，后件的出现概率。

一般来说，要找到关联规则，我们需要先根据支持度阈值，产生所有频繁项集；然后根据置信度阈值，在它们中提取所有高置信度的规则。

但如此一来，候选项集的数量就十分巨大。Apriori 算法就用于减少候选项集的数量。它的原理有两条：

1. 如果一个项集是频繁的，则它的所有子集一定也是频繁的。
2. 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。

由此，我们就可以进行大量的剪枝，从而降低计算量。

五、实验目的：

1. 学会调用 weka 包实现关联规则的挖掘。
2. 自己编程实现 Apriori 算法。

六、实验内容：

1. 使用 weka 的 GUI 实现关联规则挖掘。

2. 实现 Apriori 算法。

七、实验器材（设备、元器件）：

PC 微机一台。

八、实验步骤：

核心代码：

```
public void run() {
    // Record all frequent item sets.
    List<Map<Set<String>, Double>> frequentItemSetsList = new ArrayList<Map<Set<String>, Double>>();

    // Start with frequent 1 item sets.
    frequentItemSetsList.add(getFrequentOneItemSets());

    while (true) {
        // Get the frequent k-1 item sets.
        Map<Set<String>, Double> frequentLastItemSets = frequentItemSetsList.get(frequentItemSetsList.size() - 1);

        // Get the frequent k item sets.
        Map<Set<String>, Double> frequentKItemSets = getFrequentKItemSets(frequentLastItemSets);

        // If there is no frequent k item set, stop exploring.
        if (frequentKItemSets.isEmpty()) {
            break;
        }

        // Otherwise, record the frequent k item sets.
        frequentItemSetsList.add(frequentKItemSets);
    }

    // Flatten all frequent item sets into a single map.
    Map<Set<String>, Double> allFrequentItemSets = new HashMap<Set<String>, Double>();
    for (Map<Set<String>, Double> frequentItemSets : frequentItemSetsList) {
        allFrequentItemSets.putAll(frequentItemSets);
    }

    // Print all frequent item sets.
    IOUtils.printFrequentItemSets(allFrequentItemSets);

    // Get all association rules.
    Map<Map<Set<String>, Set<String>>, Double> associationRules = getAssociationRules(allFrequentItemSets);

    // Print all association rules.
    IOUtils.printAssociationRules(associationRules);
}
```

获取频繁一项集：

```

private Map<Set<String>, Double> getFrequentOneItemSets() {
    // Record all frequent 1 item sets, and their corresponding support values.
    Map<Set<String>, Double> frequentOneItemSets = new HashMap<Set<String>, Double>();

    // Record all 1 item sets that have been counted, to avoid duplicate counting.
    Set<Set<String>> countedOneItemSets = new HashSet<Set<String>>();

    for (Set<String> itemSet : itemSets) {
        for (String item : itemSet) {
            // Treat each single item as a 1 item set.
            Set<String> oneItemSet = new HashSet<String>();
            oneItemSet.add(item);

            // If this 1 item set has been counted, skip it.
            if (countedOneItemSets.contains(oneItemSet)) {
                continue;
            }

            // If this 1 item set is frequent, record it.
            double support = getSupport(oneItemSet);
            if (support ≥ minSupport) {
                frequentOneItemSets.put(oneItemSet, support);
            }

            // Mark this 1 item set as counted.
            countedOneItemSets.add(oneItemSet);
        }
    }

    return frequentOneItemSets;
}

```

获取频繁 K 项集:

```

private Map<Set<String>, Double> getFrequentKItemSets(Map<Set<String>, Double> frequentLastItemSets) {
    // Record all frequent k item sets, and their corresponding support values.
    Map<Set<String>, Double> frequentKItemSets = new HashMap<Set<String>, Double>();

    // Record all k item sets that have been counted, to avoid duplicate counting.
    Set<Set<String>> countedKItemSets = new HashSet<Set<String>>();

    // For every two k-1 frequent item sets.
    for (Set<String> frequentLastItemSet1 : frequentLastItemSets.keySet()) {
        for (Set<String> frequentLastItemSet2 : frequentLastItemSets.keySet()) {
            // If the two k-1 item sets are the same, skip it.
            if (frequentLastItemSet1.equals(frequentLastItemSet2)) {
                continue;
            }

            // Merge the two k-1 item sets into a new item set.
            Set<String> kItemSet = new HashSet<String>(frequentLastItemSet1);
            kItemSet.addAll(frequentLastItemSet2);

            // If the new item set is not a k item set, skip it.
            if (kItemSet.size() ≠ frequentLastItemSet1.size() + 1) {
                continue;
            }

            // If the k item set has been counted, skip it.
            if (countedKItemSets.contains(kItemSet)) {
                continue;
            }

            // If the k item set has an infrequent k-1 subset, skip it.
            if (hasInfrequentSubset(kItemSet, frequentLastItemSets)) {
                continue;
            }

            // If the merged item set is frequent, record it.
            double support = getSupport(kItemSet);
            if (support ≥ minSupport) {
                frequentKItemSets.put(kItemSet, support);
            }

            // Mark this k item set as counted.
            countedKItemSets.add(kItemSet);
        }
    }

    return frequentKItemSets;
}

```

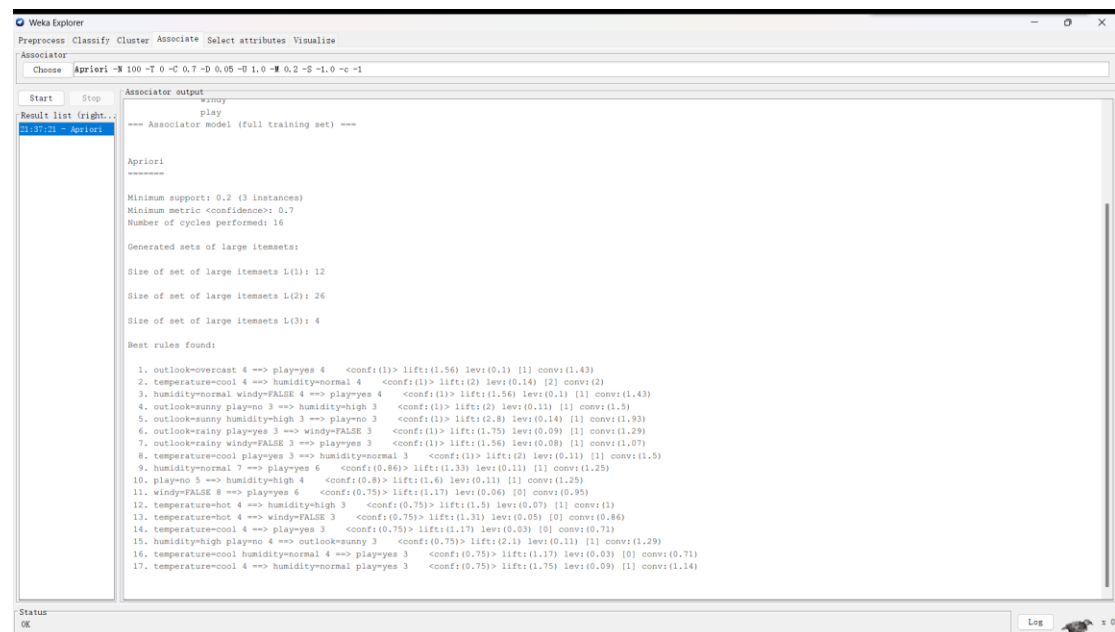
判断子集是否频繁：

```
private static boolean hasInfrequentSubset(Set<String> itemSet, Map<Set<String>, Double> frequentLastItemSets) {
    // For every k-1 subset of the given item set.
    for (String item : itemSet) {
        Set<String> subset = new HashSet<String>(itemSet);
        subset.remove(item);

        // If this subset is not frequent, return true.
        if (!frequentLastItemSets.containsKey(subset)) {
            return true;
        }
    }
    return false;
}
```

九、实验数据及结果分析：

使用 weka 对 weather.arff 进行关联规则挖掘，令支持度为 0.2，置信度为 0.7，挖掘出 17 条规则。



使用 Java 对 top1000data.txt 进行关联规则挖掘，令支持度为 0.1，置信度为 0.5，挖掘出 7 条规则。

```
All frequent item sets:
[48, 41], support=0.13
[39, 41], support=0.19
[48], support=0.45
[38], support=0.24
[39], support=0.61
[48, 39, 41], support=0.11
[48, 38], support=0.11
[48, 39], support=0.33
[38, 39], support=0.15
[41], support=0.24
[32], support=0.12

All association rules:
[48] => [39], confidence=0.74
[39] => [48], confidence=0.54
[38] => [39], confidence=0.62
[48, 41] => [39], confidence=0.84
[39, 41] => [48], confidence=0.59
[41] => [48], confidence=0.56
[41] => [39], confidence=0.8
```

十、实验结论：

编程实现的 Apriori 算法在分析数据文件时，能够正确完成自连接、剪枝等操作；在支持度 0.1、置信度 0.5 的条件下，挖掘出了 7 条规则。

十一、总结及心得体会：

通过本实验，我对关联规则挖掘的原理有了进一步的认识，充分理解了 Apriori 算法剪枝的核心思想，对于编程实现 Apriori 算法有了新的心得。

十二、对本实验过程及方法、手段的改进建议：

无。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：蔡与望 学号：2020010801024 指导教师：蔡世民

实验地点：主楼 A2-413A

实验时间：2022/11/29

一、实验室名称：计算机学院实验中心

二、实验项目名称：分类

三、实验学时：2 学时

四、实验原理：

KNN 和决策树是经典的两种分类算法。

KNN 是一种思路简单、易于实现的分类算法。它根据距离函数计算待分类样本 X 和每个训练样本的距离（作为相似度），选择与待分类样本距离最小的 K 个样本作为 X 的 K 个最近邻，最后以 X 的 K 个最近邻中的大多数所属的类别作为 X 的类别。

感知机是典型的监督学习。在训练期间，不断用训练集中的每个模式对训练网络。当给定某一训练模式时，感知机输出单元会输出一个实际输出向量，用期望输出与实际输出之差来修正网络连接权值。权值的更新采用最简单的 δ 学习规则，那么，感知机学习算法为：

$$W_{ij}(k+1) = W_{ij}(k) + \eta[t_i - a_i(k)]p_j(k)$$

BP 是通过连续不断地在相对于误差函数斜率下降的方向上计算网络权值和偏差的变化而逐渐逼近目标的。每一次权值和偏差的变化都与网络误差的影响成正比，并以反向传播的方式传递到每一层。

决策树以代表训练样本的单个结点开始。如果样本都在同一个类，则该结点成为树叶，并用该类标记。否则，算法选择最有分类能力的属性作为决策树的当前结点。根据当前决策结点属性取值的不同，将训练样本数据集分为若干子集，每个取值形成一个分枝，有几个取值形成几个分枝。针对上一步得到的一个子集，重复进行先前步骤，递归形成每个划分样本上的决策树。一旦一个属性出现在一

个结点上，就不必在该结点的任何后代考虑它。

五、实验目的：

熟悉并掌握 KNN、感知机、BP 与决策树的原理和编程实现。

六、实验内容：

1. 实现 KNN 算法。
2. 初步学习 MATLAB 实现 ANN 算法。
3. 实现决策树算法（可选或课下）。

七、实验器材（设备、元器件）：

PC 微机一台。

八、实验步骤：

1. KNN

获取最近的 k 个数据：

```
private List<Iris> getKClosestIrises(Iris iris) {
    // Store the k closest irises.
    // The closest iris is at the tail of the queue.
    PriorityQueue<Neighbor> neighbors = new PriorityQueue<Neighbor>(k, new Comparator<Neighbor>() {
        @Override
        public int compare(Neighbor o1, Neighbor o2) {
            return -Double.compare(o1.getDistance(), o2.getDistance());
        }
    });

    // Find the k closest irises.
    for (Iris trainIris : trainIrises) {
        double distance = trainIris.getDistance(iris);

        // If the queue is not full, add the neighbor.
        if (neighbors.size() < k) {
            neighbors.add(new Neighbor(trainIris, distance));
            continue;
        }

        // If the neighbor is farther, skip it.
        if (distance >= neighbors.peek().getDistance()) {
            continue;
        }

        // Otherwise, remove the farthest neighbor and add this one.
        neighbors.poll();
        neighbors.add(new Neighbor(trainIris, distance));
    }

    // Transform queue to list.
    List<Iris> kClosestIrises = new ArrayList<Iris>();
    while (!neighbors.isEmpty()) {
        kClosestIrises.add(neighbors.poll().getIris());
    }

    return kClosestIrises;
}
```

在这 k 个数据中，找到出现最频繁的标签：


```

private static String getMostFrequentLabel(List<Iris> irises) {
    // Count the number of each label.
    Map<String, Integer> labelCount = new HashMap<String, Integer>();
    for (Iris iris : irises) {
        String label = iris.getLabel();
        if (labelCount.containsKey(label)) {
            labelCount.put(label, labelCount.get(label) + 1);
        } else {
            labelCount.put(label, 1);
        }
    }

    // Find the most frequent label.
    int maxCount = 0;
    String mostFrequentLabel = "";
    for (String label : labelCount.keySet()) {
        int count = labelCount.get(label);
        if (count <= maxCount) {
            continue;
        }
        maxCount = count;
        mostFrequentLabel = label;
    }

    return mostFrequentLabel;
}

```

2. ANN

感知器:

```

close all;
clc;

P = [-0.5 -0.5 0.3 0; -0.5 0.5 -0.5 1];
T = [1 1 0 0];
plotpv(P, T);

network = newp([-1 1; 0 1], 1);
network = init(network);
y = sim(network, P);
e = T - y;
w = network.IW{1, 1};
b = network.b{1};
plotpc(w, b);

while (mae(e) > 0.0015)
    dw = learnnp(w, P, [], [], [], [], e, [], [], [], [], []);
    db = learnnp(b, ones(1, 4), [], [], [], [], e, [], [], [], [], []);
    w = w + dw;
    b = b + db;
    network.IW{1, 1} = w;
    network.b{1} = b;
    plotpc(w, b);
    pause;
    y = sim(network, P);
    e = T - y;
end

```

反向传播：

```
clear all;
close all;
clc;
warning off;

p = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];

network = newff([0 8], [10 1], {'tansig' 'purelin'}, 'trainlm');
y1 = sim(network, p);
plot(p, t, 'o', p, y1, 'x');

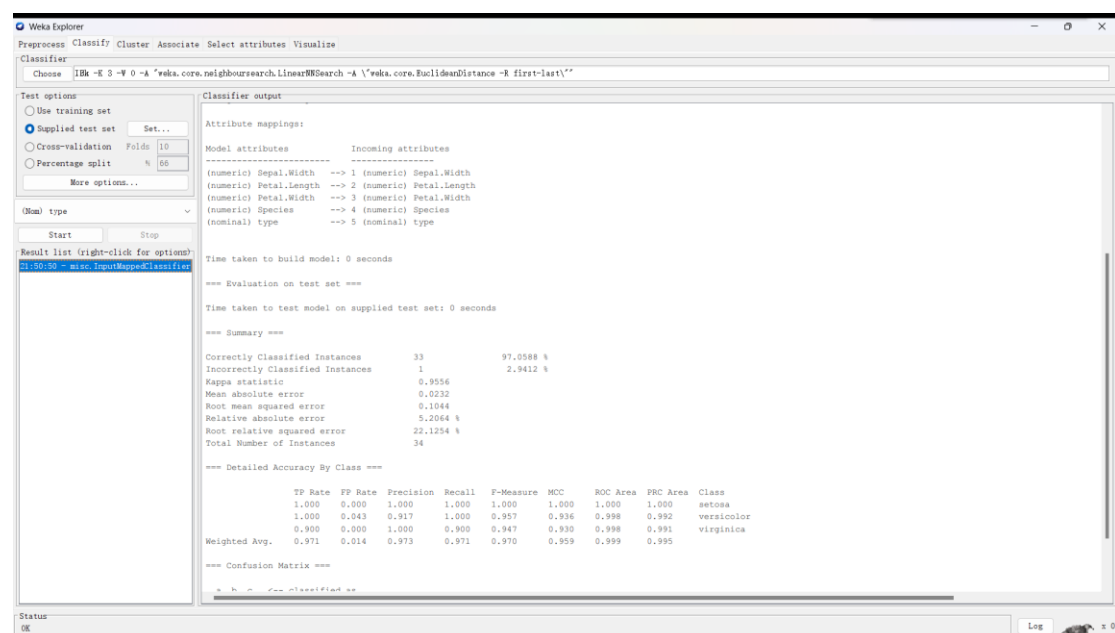
network.trainParam.epochs = 50;
network.trainParam.goal = 0.01;
network = train(network, p, t);
y2 = sim(network, p);

test = 6.5;
y3 = sim(network, test);
plot(p, t, 'o', p, y1, 'x', p, y2, '*');
```

九、实验数据及结果分析：

1. KNN

使用 weka 对 iris-test.csv 进行分类，准确率 97.05%，错误 1 次。



The screenshot shows the Weka Explorer interface with the 'Classifier' tab selected. The 'Test options' section is set to 'Supplied test set'. The 'Classifier output' pane displays the following information:

Attribute mappings:

Model attributes	Incoming attributes
(numeric) Sepal.Width	--> 1 (numeric) Sepal.Width
(numeric) Petal.Length	--> 2 (numeric) Petal.Length
(numeric) Petal.Width	--> 3 (numeric) Petal.Width
(numeric) Species	--> 4 (numeric) Species
(nominal) type	--> 5 (nominal) type

Time taken to build model: 0 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correctly Classified Instances	33	97.058 %
Incorrectly Classified Instances	1	2.9412 %
Kappa statistic	0.9556	
Mean absolute error	0.0232	
Root mean squared error	0.1044	
Relative absolute error	5.2064 %	
Root relative squared error	22.1254 %	
Total Number of Instances	34	

=== 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	1.000	setosa
1.000	0.043	0.917	1.000	0.957	0.936	0.998	0.992	0.992	versicolor
0.900	0.000	1.000	0.900	0.900	0.947	0.930	0.998	0.991	virginica
Weighted Avg.	0.971	0.014	0.973	0.971	0.970	0.959	0.999	0.995	

=== Confusion Matrix ===

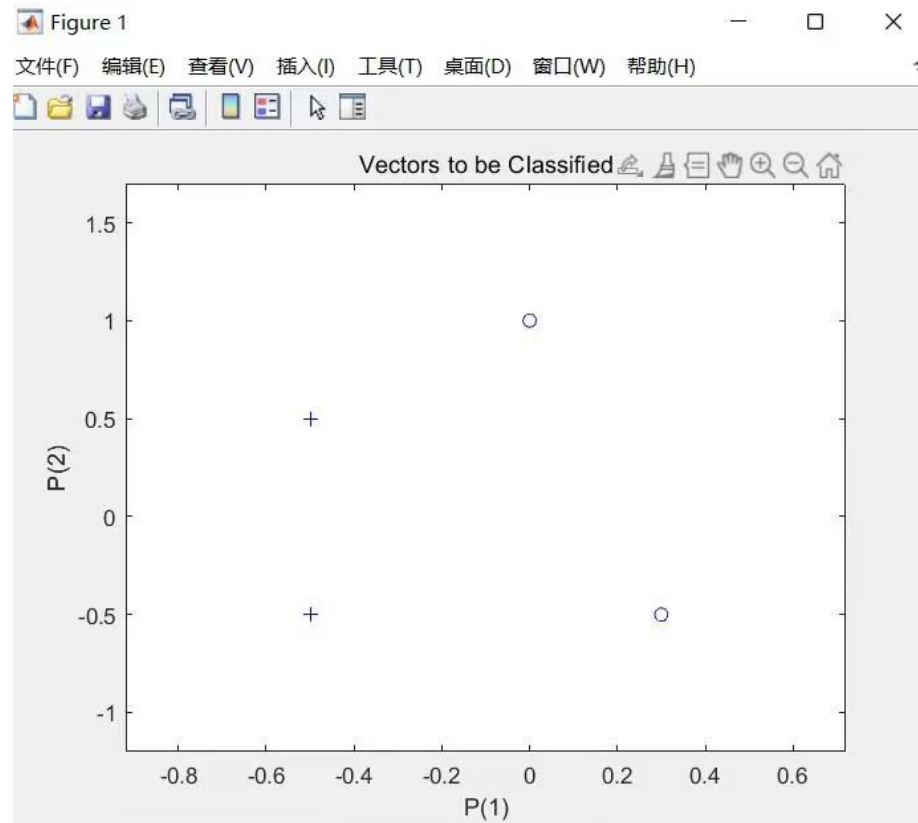
	Actual \ Predicted	setosa	versicolor	virginica
Predicted setosa	33	0	0	0
Predicted versicolor	0	0	33	1
Predicted virginica	0	0	0	34

使用 Java 对 iris-test.csv 进行分类，准确率 97.05%，错误 1 次。

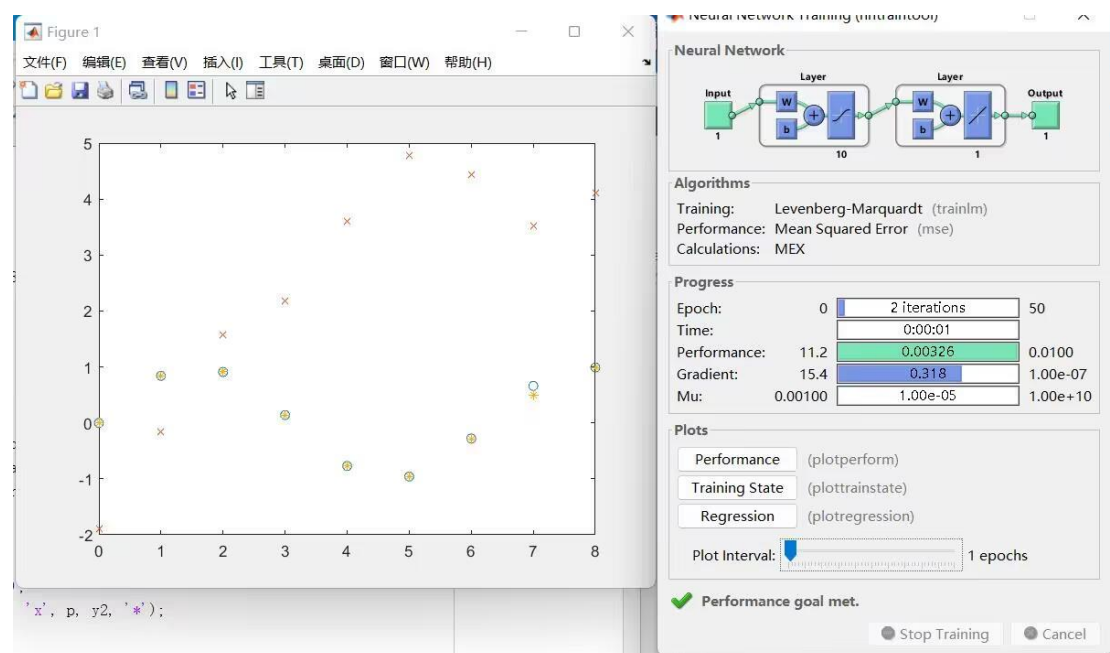
[4.9, 2.5, 4.5, 1.7] => virginica prediction=versicolor
Accuracy: 97.05%

2. ANN

感知机结果：



BP 结果：



十、实验结论：

KNN 实现了较为准确的分类，在测试集上取得了 97.05%的准确率。感知器和 BP 的训练效果良好。

十一、总结及心得体会：

通过本次实验，我对数据挖掘中分类的概念有了进一步的认识，对于 KNN、感知机和 BP 的原理和应用有了亲身的操作和实践。

十二、对本实验过程及方法、手段的改进建议：

无。

报告评分：

指导教师签字：

电子科技大学

实验报告

学生姓名：蔡与望 学号：2020010801024 指导教师：蔡世民

实验地点：主楼 A2-413A

实验时间：2022/12/6

一、实验室名称：计算机学院实验中心

二、实验项目名称：聚类

三、实验学时：2 学时

四、实验原理：

聚类就是将数据分为多个簇，使得在同一个簇内对象之间具有较高的相似度，而不同簇之间的对象差别较大。聚类分析是获得数据内部结构的有效方法。通过观察聚类得到的每个簇的特点，可以集中对特定的某些簇作进一步分析。

K 均值和 DBSCAN 是聚类的两种方法。

K 均值的目标是将数据集 D 划分为 K 个互不相容的簇，使得簇内对象互相相似，簇间差异大。聚类目标函数：簇对象到簇中心平方误差最小。

DBSCAN 将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇。对于一个类中的每个对象，在其给定半径的领域中包含的对象不能少于某一给定的最小数目。发现一个类的过程是基于这样的事实：一个类能够被其中的任意一个核心对象所确定。

五、实验目的：

熟悉并掌握 K 均值和 DBSCAN 的原理和编程实现。

六、实验内容：

1. 学会调用 WEKA 包和自己编程实现 Kmeans 聚类算法；
2. 学会调用 WEKA 包和自己编程实现 DBSCAN 聚类算法。

七、实验器材（设备、元器件）：

PC 微机一台。

八、实验步骤:

1. K means

核心代码:

```
private void cluster() {
    while (true) {
        List<Point> nextCenters = getNextCenters();
        if (isSamePoints(centers, nextCenters)) {
            break;
        }
        centers = nextCenters;
    }
}
```

获取下一次迭代的中心:

```
private List<Point> getNextCenters() {
    // Store the next centers' coordinates.
    List<Point> nextCenters = new ArrayList<Point>();
    for (int centerIndex = 0; centerIndex < centers.size(); centerIndex++) {
        nextCenters.add(new Point(0, 0));
    }

    // Record the number of points in each next cluster.
    int clusterNum = centers.size();
    int[] pointNums = new int[clusterNum];

    for (Point point : points) {
        // Find the closest center.
        int closestCenterIndex = getClosestCenterIndex(point);

        // Assign the point to the cluster.
        point.setClusterIndex(closestCenterIndex);
        pointNums[closestCenterIndex]++;

        // Update the center's coordinates.
        // The current coordinates are the sum of all points' coordinates
        // in the cluster, used to calculate the mean value later.
        Point nextCenter = nextCenters.get(closestCenterIndex);
        nextCenter.setX(nextCenter.getX() + point.getX());
        nextCenter.setY(nextCenter.getY() + point.getY());
    }

    // Calculate the coordinates of the next centers.
    for (int clusterIndex = 0; clusterIndex < clusterNum; clusterIndex++) {
        // If there is no point in the cluster, spawn this center randomly again.
        if (pointNums[clusterIndex] == 0) {
            nextCenters.set(clusterIndex, createRandomPoint());
            continue;
        }

        // Otherwise, calculate the mean of the cluster, and treat it as the new center.
        Point nextCenter = nextCenters.get(clusterIndex);
        nextCenter.setX(nextCenter.getX() / pointNums[clusterIndex]);
        nextCenter.setY(nextCenter.getY() / pointNums[clusterIndex]);
    }

    return nextCenters;
}
```

2. DBSCAN

核心代码:

```
private void cluster() {
    for (Point point : points) {
        // If this point is already visited, skip it.
        if (point.getIsVisited()) {
            continue;
        }

        // If this point is not a center, don't visit it.
        // It is either a noise point or an edge point of a certain cluster.
        List<Point> neighbors = getNeighbors(point);
        if (neighbors.size() < minNeighborNum) {
            continue;
        }

        // Otherwise, start a new cluster.
        currentClusterIndex++;

        // Add this point to the cluster, and explore all of its neighbors.
        point.setIsVisited(true);
        point.setClusterIndex(currentClusterIndex);
        for (Point neighbor : neighbors) {
            exploreCluster(neighbor);
        }
    }
}
```

DFS 探索当前簇内点:

```
private void exploreCluster(Point point) {
    // If this point is already visited, skip it.
    if (point.getIsVisited()) {
        return;
    }

    // Add this point to the cluster.
    point.setIsVisited(true);
    point.setClusterIndex(currentClusterIndex);

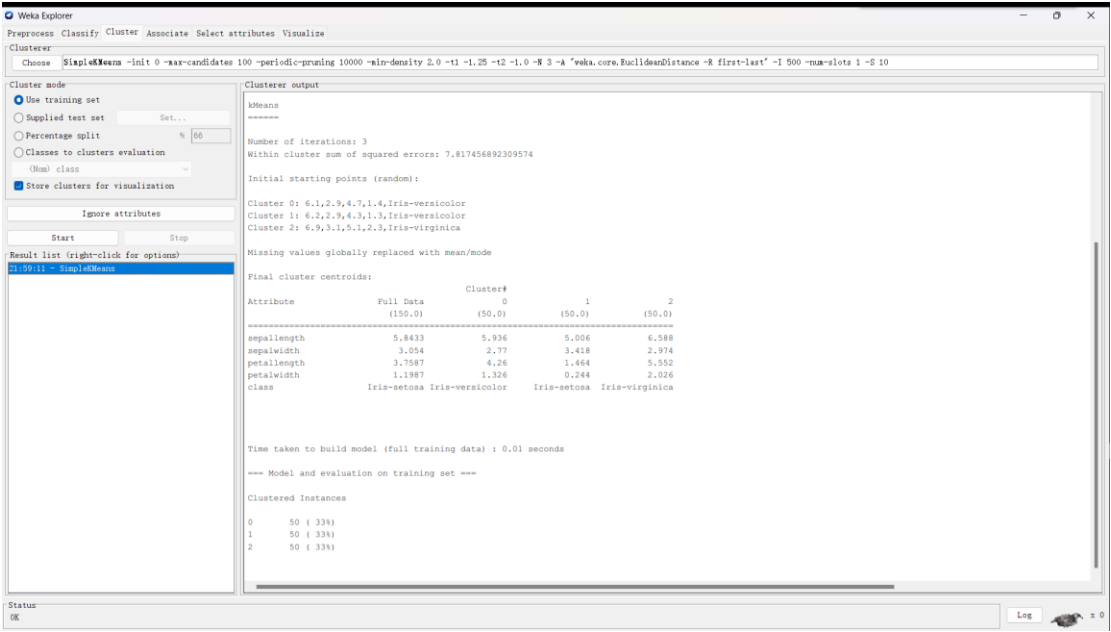
    // If this point is not a center, stop further exploration from this point.
    List<Point> neighbors = getNeighbors(point);
    if (neighbors.size() < minNeighborNum) {
        return;
    }

    // Otherwise, continue to explore all of its neighbors.
    for (Point neighbor : neighbors) {
        exploreCluster(neighbor);
    }
}
```

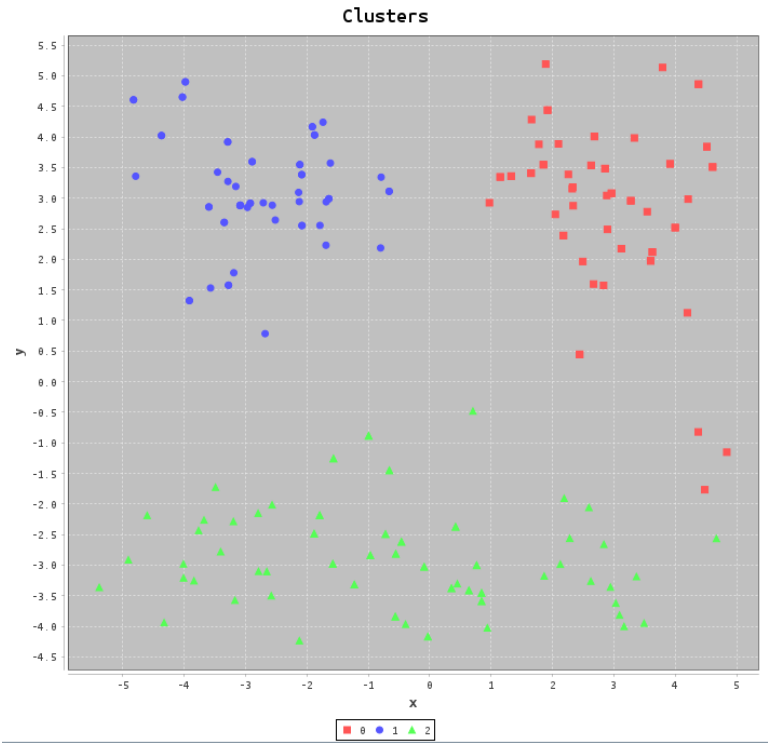
九、实验数据及结果分析:

1. K means

使用 weka 对 iris.arff 进行聚类，数据集被分为 3 个簇，每簇各有 50 项：

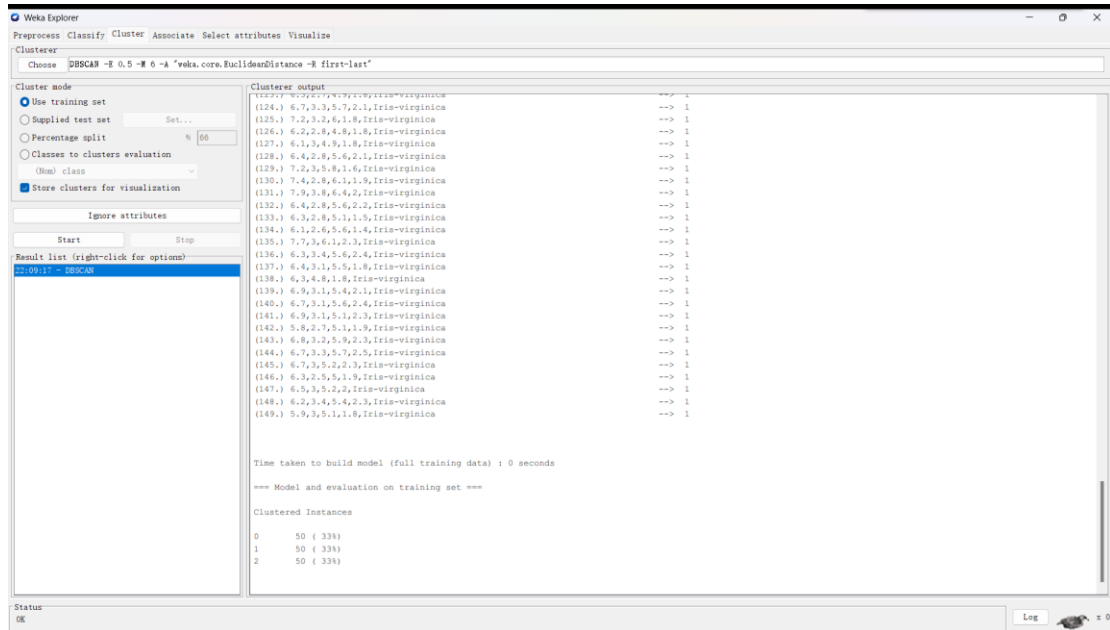


使用 Java 对 points.txt 进行聚类，绘制出各簇分布图，数据集被分为 3 个簇。

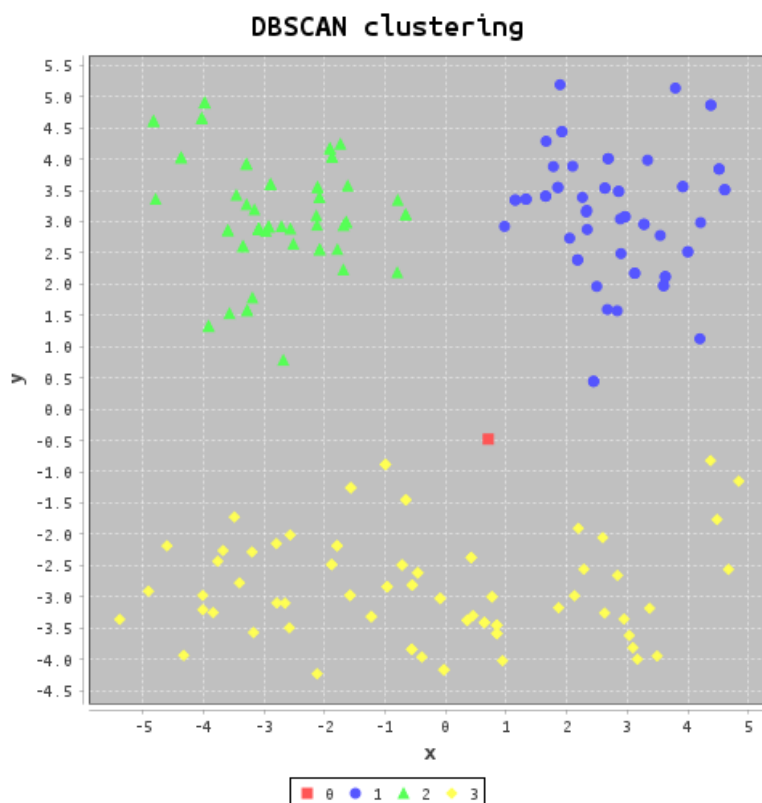


2. DBSCAN

使用 weka 对 iris.arff 进行聚类，令 eps 为 0.5，簇内最小点数为 6，数据集被分为 3 个簇，各簇有 50 个数据。



使用 Java 对 points.txt 进行聚类，令 eps 为 1.5，簇内最小点数为 2，分出了 3 个簇，1 个噪声点：



十、实验结论：

K means 可以简单、快速地分簇，在本例中分出了 3 个簇，但对噪声的抵抗力低。DBSCAN 因为每加入一个点都要判断是否为核心，性能和易读性都较

差，但可以检测出噪声。本例中，DBSCAN 分出了 3 个簇，并相比于 Kmeans 额外检测出了一个噪声点。

十一、总结及心得体会：

通过本次实验，我对数据挖掘中聚类的概念有了进一步的认识，对于 K 均值和 DBSCAN 的原理和应用有了亲身的操作和实践。

十二、对本实验过程及方法、手段的改进建议：

无。

报告评分：

指导教师签字：