# 机器学习及应用实验报告

## ——WEKA API 的熟悉与应用

| | | |
|---|---|---|
| 实验编号 | ： | 二 |
| 实验名称 | ： | WEKA API 的熟悉与应用 |
| 实验室 | ： | 电光学院 105 |
| 姓名 | ： | 刘嘉洋 |
| 学号 | ： | 1412620 |
| 班级 | ： | 软件学院 14 级二班 |
| 实验日期 | ： | 2017 年 3 月 7 日 |
| 至 | ： | 2017 年 3 月 14 日 |

| | |
|---|---|
| 评分教师 | 实验报告成绩 |
| 评分日期 | 年　　月　　日 |

# 一、实验目的

- 在 Eclipse 中进行 weka.jar 的配置
- 结合 weka-src.jar 及说明文档进行 API 的学习
- 回顾 WEKA 基本模块及应用
- 完成 Instances,Filters,Classifier,Evaluation 类的编程操作
- 对比分析常见分类器的表现结果
- 掌握便捷获取.arff 格式文件的技巧

# 二、实验环境

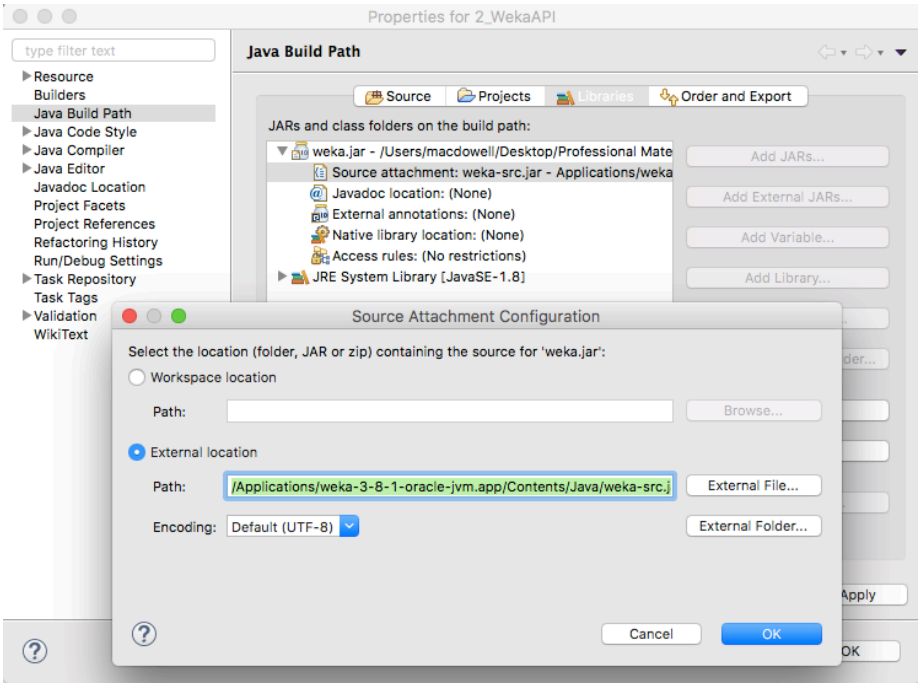
- macOS Sierra Version 10.12
- weka-3-8-1
- Eclipse Mars Release 4.5.0

# 三、实验报告

1. Instance 类 • 数据读取与统计输出

step1: 在 Eclipse 中新建 Java Project

step2: 导入 weka.jar 并配置源码路径指向 weka-src.jar，以便 API 查阅




step3: 新建类，调用 getFileInstance 方法指向 bank.arff 数据集，获取数据实例

```
Instances instances = getFileInstances(
        "/Users/macdowell/Desktop/Professional Materials/3_Junior_S2/2_MachineLearning/Lab/2_WekaAPI/实验二 数据集/bank.arff");
```

step4: [重要] 调用实例对象的 setClassIndex 方法，将最后一列设置为类别

```
// 注意：要首先设置类属性标签，不然会在调用numClasses()时报"UnassignedClassException"
instances.setClassIndex(instances.numAttributes() - 1); // 设置最后一列为类别
```

step5: 调用对应方法，分别获取数据集样例数、属性数、类别数，并打印输出

```
System.out.println("样例数: " + instances.numInstances());
System.out.println("属性数: " + instances.numAttributes());
System.out.println("类别数: " + instances.numClasses());
```

step6: InstanceAssign 类控制台运行结果

```
样例数: 600
属性数: 11
类别数: 2
```

## 2. Filters 类 • 属性操作

### 2.1 Discretize 类 • 离散化

step1: 新建类，重复上述操作完成数据实例初始化 (bank.arff)

step2: 自定义属性离散化方法 discretizeAttribute

其中：option 取”-R”表示对整列进行离散化，并在紧接着的 option 指定目标属性列号，列号从 1 开始，多列则标号间以’,’间隔

```
/**
 * 属性离散化方法
 * @param instances 数据实例
 * @throws Exception
 */
public static void discretizeAttribute(Instances instances) throws Exception {
    // 离散化第1、4个属性
    Discretize discretize = new Discretize();
    String[] options = new String[2];
    options[0] = "-R";  // Specifies list of columns to Discretize.
    options[1] = "1,4"; // 指定多列 中间以,间隔
    // options[2] = "-R"; // 这样的写法是不对的
    // options[3] = "4";
    discretize.setOptions(options);
    discretize.setInputFormat(instances);
    Instances newData = Filter.useFilter(instances, discretize);
    for (int i = 0; i < 20; i++) {
        // instance( i )是得到第i个样本
        System.out.println(newData.instance(i));
    }
}
```

step3: main 方法中调用 discretizeAttribute，控制台运行结果

```
-----------------离散化属性-----------------
'\'(47.4-52.3]\'',FEMALE,INNER_CITY,'\'(16637.388-22448.977]\'',NO,1,NO,NO,NO,NO,YES
'\'(37.6-42.5]\'',MALE,TOWN,'\'(28260.566-34072.155]\'',YES,3,YES,NO,YES,YES,NO
'\'(47.4-52.3]\'',FEMALE,INNER_CITY,'\'(10825.799-16637.388]\'',YES,0,YES,YES,YES,NO,NO
'\'(22.9-27.8]\'',FEMALE,TOWN,'\'(16637.388-22448.977]\'',YES,3,NO,NO,YES,NO,NO
'\'(52.3-57.2]\'',FEMALE,RURAL,'\'(45695.333-51506.922]\'',YES,0,NO,YES,NO,NO,NO
'\'(52.3-57.2]\'',FEMALE,TOWN,'\'(34072.155-39883.744]\'',YES,2,NO,YES,YES,NO,YES
'\'(-inf-22.9]\'',MALE,RURAL,'\'(-inf-10825.799]\'',NO,0,NO,NO,YES,NO,YES
'\'(57.2-62.1]\'',MALE,TOWN,'\'(22448.977-28260.566]\'',YES,0,YES,YES,YES,NO,NO
'\'(32.7-37.6]\'',FEMALE,SUBURBAN,'\'(22448.977-28260.566]\'',YES,2,YES,NO,NO,NO,NO
'\'(52.3-57.2]\'',MALE,TOWN,'\'(22448.977-28260.566]\'',YES,2,YES,YES,YES,NO,NO
'\'(62.1-inf)\'',FEMALE,TOWN,'\'(57318.511-inf)\'',YES,0,NO,YES,YES,NO,NO
'\'(47.4-52.3]\'',FEMALE,INNER_CITY,'\'(22448.977-28260.566]\'',NO,0,YES,YES,YES,YES,NO
'\'(42.5-47.4]\'',FEMALE,TOWN,'\'(10825.799-16637.388]\'',YES,1,NO,YES,YES,YES,YES
'\'(62.1-inf)\'',FEMALE,TOWN,'\'(51506.922-57318.511]\'',YES,1,YES,YES,YES,YES,YES
'\'(32.7-37.6]\'',MALE,RURAL,'\'(16637.388-22448.977]\'',YES,0,NO,YES,YES,YES,NO
'\'(37.6-42.5]\'',FEMALE,INNER_CITY,'\'(16637.388-22448.977]\'',YES,0,YES,YES,YES,YES,NO
'\'(32.7-37.6]\'',FEMALE,TOWN,'\'(16637.388-22448.977]\'',YES,2,NO,NO,NO,YES,NO
'\'(42.5-47.4]\'',FEMALE,SUBURBAN,'\'(39883.744-45695.333]\'',YES,0,NO,YES,NO,YES,NO
'\'(57.2-62.1]\'',FEMALE,INNER_CITY,'\'(22448.977-28260.566]\'',YES,0,NO,YES,NO,NO,YES
'\'(27.8-32.7]\'',MALE,TOWN,'\'(22448.977-28260.566]\'',YES,0,YES,YES,YES,NO,NO
```

## 2.2 Normalize 类 • 归一化

step4: 自定义属性归一化方法 normalizeAttribute

其中：option 取”-T”：归一区间的起始坐标

取”-S”：归一区间的长度

[注] Normalize 会自动归一化所有 numeric 类型的属性

```
/**
 * 属性归一化方法
 * @param instances 数据实例
 * @throws Exception
 */
public static void normalizeAttribute(Instances instances) throws Exception {
    Normalize normalize = new Normalize();
    String[] options = new String[4];
    // 归一化所有numeric类型的属性
    // 归一区间设置为[-1,1]
    options[0] = "-T"; // The translation of the output range.
    options[1] = "-1"; // 从-1起始
    options[2] = "-S"; // The scaling factor for the output range.
    options[3] = "2"; // 跨度为2
    normalize.setOptions(options);
    normalize.setInputFormat(instances);
    Instances newData = Filter.useFilter(instances, normalize);
    for (int i = 0; i < 20; i++) {
        System.out.println(newData.instance(i));
    }
}
```

step5: main 方法中调用 normalizeAttribute，控制台运行结果

```
------------------归一化属性------------------
0.22449,FEMALE,INNER_CITY,-0.568731,NO,1,NO,NO,NO,NO,YES
-0.102041,MALE,TOWN,-0.13721,YES,3,YES,NO,YES,YES,NO
0.346939,FEMALE,INNER_CITY,-0.602133,YES,0,YES,YES,YES,NO,NO
-0.795918,FEMALE,TOWN,-0.47136,YES,3,NO,NO,YES,NO,NO
0.591837,FEMALE,RURAL,0.567974,YES,0,NO,YES,NO,NO,NO
0.591837,FEMALE,TOWN,0.130685,YES,2,NO,YES,YES,NO,YES
-0.836735,MALE,RURAL,-0.867064,NO,0,NO,NO,YES,NO,YES
0.632653,MALE,TOWN,-0.314047,YES,0,YES,YES,YES,NO,NO
-0.22449,FEMALE,SUBURBAN,-0.301737,YES,2,YES,NO,NO,NO,NO
0.469388,MALE,TOWN,-0.339324,YES,2,YES,YES,YES,NO,NO
0.959184,FEMALE,TOWN,0.885532,YES,0,NO,YES,YES,NO,NO
0.387755,FEMALE,INNER_CITY,-0.255123,NO,0,YES,YES,YES,YES,NO
0.061224,FEMALE,TOWN,-0.631027,YES,1,NO,YES,YES,YES,YES
0.959184,FEMALE,TOWN,0.727255,YES,1,YES,YES,YES,YES,YES
-0.265306,MALE,RURAL,-0.50236,YES,0,NO,YES,YES,YES,NO
-0.183673,FEMALE,INNER_CITY,-0.403678,YES,0,YES,YES,YES,YES,NO
-0.22449,FEMALE,TOWN,-0.562406,YES,2,NO,NO,NO,YES,NO
0.142857,FEMALE,SUBURBAN,0.238965,YES,0,NO,YES,NO,YES,NO
0.795918,FEMALE,INNER_CITY,-0.246506,YES,0,NO,YES,NO,NO,YES
-0.469388,MALE,TOWN,-0.397459,YES,0,YES,YES,YES,NO,NO
```

## 2.3 Remove 类 • 删除

step6: 自定义属性删除方法 removeAttribute

其中：option 取”-R” 表示对整列进行删除，并在紧接着的 option 指定目标属性列号，列号从 1 开始，多列则标号间以’,’间隔

[注] Remove 的删除只是将程序中处理的数据集进行过滤，并不会对.arff 进行改变

```java
/**
 * 属性删除方法
 * @param instances 数据实例
 * @throws Exception
 */
public static void removeAttribute(Instances instances) throws Exception {
    Remove remove = new Remove();
    String[] options = new String[2];
    // 删除第2、3个属性
    options[0] = "-R";
    options[1] = "2,3"; // 指定多列 中间以,间隔
    remove.setOptions(options);
    remove.setInputFormat(instances);
    Instances newData = Filter.useFilter(instances, remove);
    for (int i = 0; i < 20; i++) {
        System.out.println(newData.instance(i));
    }
}
```

step7: main 方法中调用 normalizeAttribute，控制台运行结果

```
------------------删除属性------------------
48,17546,NO,1,NO,NO,NO,NO,YES
40,30085.1,YES,3,YES,NO,YES,YES,NO
51,16575.4,YES,0,YES,YES,YES,NO,NO
23,20375.4,YES,3,NO,NO,YES,NO,NO
57,50576.3,YES,0,NO,YES,NO,NO,NO
57,37869.6,YES,2,NO,YES,YES,NO,YES
22,8877.07,NO,0,NO,NO,YES,NO,YES
58,24946.6,YES,0,YES,YES,YES,NO,NO
37,25304.3,YES,2,YES,NO,NO,NO,NO
54,24212.1,YES,2,YES,YES,YES,NO,NO
66,59803.9,YES,0,NO,YES,YES,NO,NO
52,26658.8,NO,0,YES,YES,YES,YES,NO
44,15735.8,YES,1,NO,YES,YES,YES,YES
66,55204.7,YES,1,YES,YES,YES,YES,YES
36,19474.6,YES,0,NO,YES,YES,YES,NO
38,22342.1,YES,0,YES,YES,YES,YES,NO
37,17729.8,YES,2,NO,NO,NO,YES,NO
46,41016,YES,0,NO,YES,NO,YES,NO
62,26909.2,YES,0,NO,YES,NO,NO,YES
31,22522.8,YES,0,YES,YES,YES,NO,NO
```

## 3. Classifier 类 • 分类器的使用与评估

### 3.1 10 折交叉验证的使用

step1: 新建类，重复上述操作完成数据实例初始化 (bank.arff)

step2: 自定义一致评估方法 unifiedEvaluation，提供给分类器对象统一的借口

```java
/**
 * 一致评估方法。评价当前数据集及分类器情况下在十折交叉认证中的正确率。
 * @param instances 数据实例
 * @param classifier 采用分类器
 * @throws Exception
 */
private static void unifiedEvaluation(Instances instances, Classifier classifier) throws Exception {
    Evaluation eval = new Evaluation(instances);
    eval.crossValidateModel(classifier, instances, 10, new Random(1)); // 使用10折交叉认证
    System.out.println(1 - eval.errorRate()); // 正确率 = 1 - 错误率
}
```

step3: 分别定义分类器并调用 unifiedEvaluation 方法

```
System.out.println("------------------使用朴素贝叶斯分类器------------------");
Classifier classifier1 = new NaiveBayes();
unifiedEvaluation(instances, classifier1);
System.out.println("\n------------------使用SMO分类器------------------");
Classifier classifier2 = new SMO();
unifiedEvaluation(instances, classifier2);
System.out.println("\n------------------使用J48分类器------------------");
Classifier classifier3 = new J48();
unifiedEvaluation(instances, classifier3);
System.out.println("\n------------------使用1NN分类器------------------");
Classifier classifier4 = new IBk();
((IBk) classifier4).setKNN(1);
unifiedEvaluation(instances, classifier4);
```

step4: ClassifierAssign1 类控制台运行结果

```
------------------使用朴素贝叶斯分类器--------------
0.69

------------------使用SMO分类器------------------
0.7083333333333333

------------------使用J48分类器------------------
0.91

------------------使用1NN分类器------------------
0.7150000000000001
```

step5: 在 WEKA 打开 bank.arff 数据集依次使用四种分类器，核对输出验证程序正确性

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         414               69      %
Incorrectly Classified Instances       186               31      %
Kappa statistic                          0.3724
Mean absolute error                      0.3773
Root mean squared error                  0.4397
Relative absolute error                 76.0306 %
Root relative squared error             88.2766 %
Total Number of Instances              600
```

a. 朴素贝叶斯

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         425               70.8333 %
Incorrectly Classified Instances       175               29.1667 %
Kappa statistic                          0.4062
Mean absolute error                      0.2917
Root mean squared error                  0.5401
Relative absolute error                 58.7714 %
Root relative squared error            108.4169 %
Total Number of Instances              600
```

b. SMO

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         546               91      %
Incorrectly Classified Instances        54                9      %
Kappa statistic                          0.8178
Mean absolute error                      0.1559
Root mean squared error                  0.2903
Relative absolute error                 31.4168 %
Root relative squared error             58.2815 %
Total Number of Instances              600
```

c. J48

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         429               71.5    %
Incorrectly Classified Instances       171               28.5    %
Kappa statistic                          0.4259
Mean absolute error                      0.2858
Root mean squared error                  0.5329
Relative absolute error                 57.5879 %
Root relative squared error            106.9734 %
Total Number of Instances              600
```

d. 1NN

图 1 WEKA 中四种分类器对 bank.arff 的表现结果

## 3.2 训练集与测试集的使用

step1: 新建类，分别对 U_segmentation_train.arff 训练集与 U_segmentation_tetst.arff 测试集进行数据实例初始化以及类别属性设置

```
String basedPath = "/Users/macdowell/Desktop/Professional Materials/3_Junior_S2/2_MachineLearning/Lab/2_WekaAPI/实验二 数据集/";
DataSource trainingSet=new DataSource( basedPath + "U_segmentation_train.arff");
Instances instancesTrain = trainingSet.getDataSet();
instancesTrain.setClassIndex(instancesTrain.numAttributes() - 1); // 设置最后一列为类别

DataSource testingSet = new DataSource( basedPath + "U_segmentation_test.arff");
Instances instanceTest = testingSet.getDataSet();
instanceTest.setClassIndex(instanceTest.numAttributes() - 1); // 设置最后一列为类别
```

step2: 自定义一致评估方法 unifiedEvaluation，提供给分类器对象统一的借口

其中：buildClassifier 方法通过训练集训练出分类模型

evaluateModel 方法通过测试集测试所得分类模型

```
/**
 * 一致评估方法。评价当前数据训练集及分类器情况下在测试集中的正确率。
 * @param instancesTrain 数据训练实例
 * @param instancesTrain 数据测试实例
 * @param classifier 采用分类器
 * @throws Exception
 */
private static void unifiedEvaluation(Instances instancesTrain, Instances instancesTest, Classifier classifier) throws Exception {
    classifier.buildClassifier(instancesTrain); // 通过训练集训练分类模型
    Evaluation eval = new Evaluation(instancesTrain);
    eval.evaluateModel(classifier, instancesTest);  // 通过测试集测试分类模型
    System.out.println(1 - eval.errorRate()); // 正确率 = 1 - 错误率
}
```

step3: ClassifierAssign2 类控制台运行结果

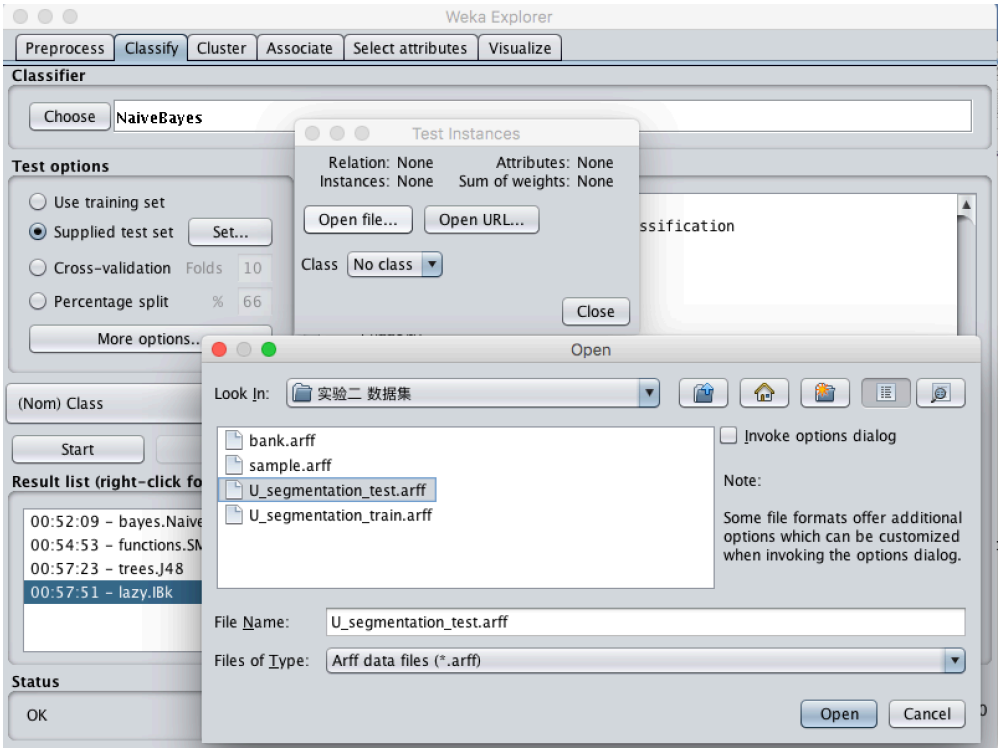
```
------------------使用朴素贝叶斯分类器--------------
0.7995238095238095

------------------使用SMO分类器------------------
0.8738095238095238

------------------使用J48分类器------------------
0.91

------------------使用1NN分类器------------------
0.9257142857142857
```

step4: 在 WEKA 输入同样的训练集和测试集使用四种分类器，核对输出验证程序正确性



```
=== Summary ===

Correctly Classified Instances        1679               79.9524 %
Incorrectly Classified Instances       421               20.0476 %
Kappa statistic                          0.7661
Mean absolute error                      0.0578
Root mean squared error                  0.2298
Relative absolute error                 23.5953 %
Root relative squared error             65.6684 %
Total Number of Instances             2100
```

a. 朴素贝叶斯

```
=== Summary ===

Correctly Classified Instances        1835               87.381  %
Incorrectly Classified Instances       265               12.619  %
Kappa statistic                          0.8528
Mean absolute error                      0.2065
Root mean squared error                  0.3051
Relative absolute error                 84.3333 %
Root relative squared error             87.18   %
Total Number of Instances             2100
```

b. SMO

```
=== Summary ===

Correctly Classified Instances        1911               91      %
Incorrectly Classified Instances       189                9      %
Kappa statistic                          0.895
Mean absolute error                      0.0308
Root mean squared error                  0.1544
Relative absolute error                 12.558  %
Root relative squared error             44.1105 %
Total Number of Instances             2100
```

c. J48

```
=== Summary ===

Correctly Classified Instances        1944               92.5714 %
Incorrectly Classified Instances       156                7.4286 %
Kappa statistic                          0.9133
Mean absolute error                      0.0284
Root mean squared error                  0.1438
Relative absolute error                 11.6129 %
Root relative squared error             41.0831 %
Total Number of Instances             2100
```

d. 1NN

图 2 WEKA 中四种分类器对 U_segmentation 训练集与测试集的表现结果

## 4. .txt->.cvs->.arff 的转换

step1: 打开 sample.txt，将辅助说明删去，并使属性数值间以英文’,’间隔

```
2017.3.1,晴,微风,正午10摄氏度,空气潮湿,适合出行
2017.3.2,多云,无风,正午6摄氏度,空气干燥,适合出行
2017.3.3,下雨,微风,正午8摄氏度,空气潮湿,不适合出行
2017.3.4,晴,大风,正午12摄氏度,空气潮湿,适合出行
2017.3.5,多云,大风,正午-3摄氏度,空气干燥,不适合出行
```

step2: 直接将文件后缀由.txt 改为.cvs


sample.csv

step3: 此时.cvs 文件使用 Excel 或 Numbers 可直接打开

[注] Excel 打开.cvs 文件可能会出现中文乱码的情况

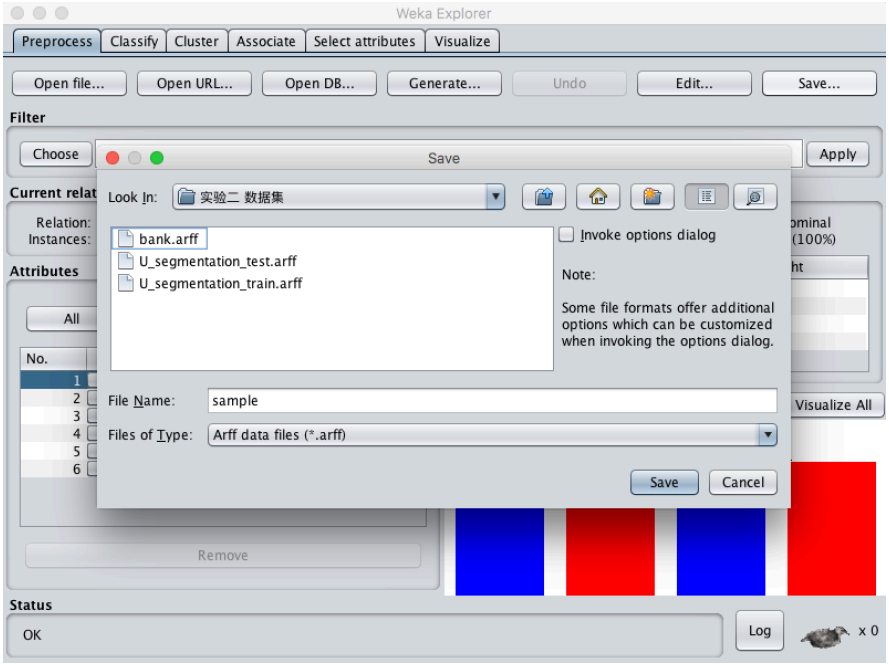
原因：.cvs 文件默认为 ANSI 编码，如果.cvs 文件的编码方式为 utf-8、Unicode 等方式时便会出现乱码

解决方法：用记事本等工具打开.cvs 文件，将其另存为并选择 ANSI 编码方式即可

Table 1

| 2017.3.1 | 晴 | 微风 | 正午10摄氏度 | 空气潮湿 | 适合出行 |
|---|---|---|---|---|---|
| 2017.3.2 | 多云 | 无风 | 正午6摄氏度 | 空气干燥 | 适合出行 |
| 2017.3.3 | 下雨 | 微风 | 正午8摄氏度 | 空气潮湿 | 不适合出行 |
| 2017.3.4 | 晴 | 大风 | 正午12摄氏度 | 空气潮湿 | 适合出行 |
| 2017.3.5 | 多云 | 大风 | 正午-3摄氏度 | 空气干燥 | 不适合出行 |

step4: 在 WEKA 中打开 sample.cvs，单击”Save…”按钮，选择.arff 保存格式




step5: 使用 Sublime 等工具打开生成的 sample.arff 文件，验证格式正确性

```
@relation sample

@attribute 2017.3.1 {2017.3.2,2017.3.3,2017.3.4,2017.3.5}
@attribute 晴 {多云,下雨,晴}
@attribute 微风 {无风,微风,大风}
@attribute 正午10摄氏度 {正午6摄氏度,正午8摄氏度,正午12摄氏度,正午-3摄氏度}
@attribute 空气潮湿 {空气干燥,空气潮湿}
@attribute 适合出行 {适合出行,不适合出行}

@data
2017.3.2,多云,无风,正午6摄氏度,空气干燥,适合出行
2017.3.3,下雨,微风,正午8摄氏度,空气潮湿,不适合出行
2017.3.4,晴,大风,正午12摄氏度,空气潮湿,适合出行
2017.3.5,多云,大风,正午-3摄氏度,空气干燥,不适合出行
```

## 5. KNN 分类器的比较与分析

step1: 在 Eclipse 工程中新建类，重复上述操作完成数据实例初始化 (bank.arff)

step2: 同样采用在上述 ClassifierAssign1 类中自定义的一致评估方法 unifiedEvaluation

step3: 构建分类器，使 K 分别取值 1、5、10、30

```
System.out.println("------------------K = 1------------------");
Classifier classifier1 = new IBk();
((IBk) classifier1).setKNN(1);
unifiedEvaluation(instances, classifier1);
System.out.println("\n------------------K = 5------------------");
Classifier classifier2 = new IBk();
((IBk) classifier2).setKNN(5);
unifiedEvaluation(instances, classifier2);
System.out.println("\n------------------K = 10------------------");
Classifier classifier3 = new IBk();
((IBk) classifier3).setKNN(10);
unifiedEvaluation(instances, classifier3);
System.out.println("\n------------------K = 30------------------");
Classifier classifier4 = new IBk();
((IBk) classifier4).setKNN(30);
unifiedEvaluation(instances, classifier4);
```

step4: KNNAssign 类控制台运行结果

```
------------------K = 1------------------
0.7150000000000001

------------------K = 5------------------
0.755

------------------K = 10-----------------
0.7783333333333333

------------------K = 30-----------------
0.7683333333333333
```

step5: 在 WEKA 打开 bank.arff 数据集使用 IBk 分类器将 K 依次取同样四种值，核对输出验证程序正确性

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         429               71.5    %
Incorrectly Classified Instances       171               28.5    %
Kappa statistic                          0.4259
Mean absolute error                      0.2858
Root mean squared error                  0.5329
Relative absolute error                 57.5879 %
Root relative squared error            106.9734 %
Total Number of Instances              600
```

a. K = 1

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         453               75.5    %
Incorrectly Classified Instances       147               24.5    %
Kappa statistic                          0.5035
Mean absolute error                      0.3415
Root mean squared error                  0.4236
Relative absolute error                 68.803  %
Root relative squared error             85.0413 %
Total Number of Instances              600
```

b. K = 5

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         467               77.8333 %
Incorrectly Classified Instances       133               22.1667 %
Kappa statistic                          0.5553
Mean absolute error                      0.3589
Root mean squared error                  0.4156
Relative absolute error                 72.3162 %
Root relative squared error             83.4228 %
Total Number of Instances              600
```

c. K = 10

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         461               76.8333 %
Incorrectly Classified Instances       139               23.1667 %
Kappa statistic                          0.5289
Mean absolute error                      0.3983
Root mean squared error                  0.4277
Relative absolute error                 80.2675 %
Root relative squared error             85.8566 %
Total Number of Instances              600
```

d. K = 30

图 3 WEKA 中 IBk 分类器四种 K 取值时的表现结果

step6: 比较与分析

①首先尝试在程序中将 K 以 1 为间隔在[1,130]连续取值找寻变化趋势

```
System.out.println("------------------K连续取值------------------");
for (int i=1; i<=130; i++) {
    Classifier classifier5 = new IBk();
    ((IBk) classifier5).setKNN(i);
    System.out.print("K = " + i + ", accuracy = ");
    unifiedEvaluation(instances, classifier5);
}
```

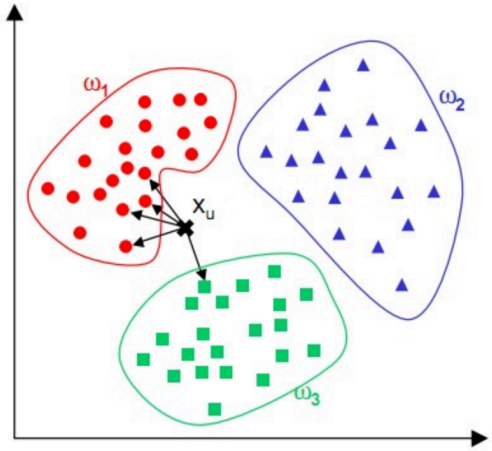
其在控制台运行结果 ([31,110]区间段没有贴出)

```
------------------K连续取值------------------
K = 1, accuracy = 0.7150000000000001
K = 2, accuracy = 0.6799999999999999
K = 3, accuracy = 0.755
K = 4, accuracy = 0.7416666666666667
K = 5, accuracy = 0.755
K = 6, accuracy = 0.7483333333333333
K = 7, accuracy = 0.7683333333333333
K = 8, accuracy = 0.775
K = 9, accuracy = 0.785
K = 10, accuracy = 0.7783333333333333
K = 11, accuracy = 0.7816666666666667
K = 12, accuracy = 0.785
K = 13, accuracy = 0.785
K = 14, accuracy = 0.7883333333333333
K = 15, accuracy = 0.78
K = 16, accuracy = 0.785
K = 17, accuracy = 0.7733333333333333
K = 18, accuracy = 0.79
K = 19, accuracy = 0.7716666666666667
K = 20, accuracy = 0.7883333333333333
K = 21, accuracy = 0.7816666666666667
K = 22, accuracy = 0.79
K = 23, accuracy = 0.785
K = 24, accuracy = 0.7916666666666666
K = 25, accuracy = 0.7766666666666666
K = 26, accuracy = 0.775
K = 27, accuracy = 0.7583333333333333
K = 28, accuracy = 0.7666666666666666
K = 29, accuracy = 0.76
K = 30, accuracy = 0.7683333333333333
```

```
K = 111, accuracy = 0.6783333333333333
K = 112, accuracy = 0.6833333333333333
K = 113, accuracy = 0.6733333333333333
K = 114, accuracy = 0.675
K = 115, accuracy = 0.6683333333333333
K = 116, accuracy = 0.6816666666666666
K = 117, accuracy = 0.6666666666666667
K = 118, accuracy = 0.6833333333333333
K = 119, accuracy = 0.6666666666666667
K = 120, accuracy = 0.6716666666666666
K = 121, accuracy = 0.6633333333333333
K = 122, accuracy = 0.665
K = 123, accuracy = 0.6616666666666666
K = 124, accuracy = 0.665
K = 125, accuracy = 0.6566666666666667
K = 126, accuracy = 0.6666666666666667
K = 127, accuracy = 0.655
K = 128, accuracy = 0.6566666666666667
K = 129, accuracy = 0.6533333333333333
K = 130, accuracy = 0.6666666666666667
```

通过观察可以得知，其准确率并没有标准的单调性，在[0.665,0.79166]间波动，在 K 取[22,24]区间段达到峰值，并在其后逐渐减小 (后测试将 K 取值区间扩大至[1,230]可确认 K 值继续增大后准确率保持着波动下降的趋势)

②回顾 KNN 算法




step1---初始化距离为最大值
step2---计算未知样本和每个训练样本的距离 dist
step3---得到目前 K 个最临近样本中的最大距离 maxdist
step4---如果 dist 小于 maxdist，则将该训练样本作为 K-最近邻样本
step5---重复步骤 2、3、4，直到未知样本和所有训练样本的距离都算完
step6---统计 K-最近邻样本中每个类标号出现的次数
step7---选择出现频率最大的类标号作为未知样本的类标号

③个人理解

最优的 K 值是取决于数据集的，其取值一方面应尽量大以使得噪点不会干扰预测的准确率，同时也不应过大避免导致 K 值主导了 KNN 算法中的其他参数反而影响准确率。在有的参考资料上看到说$\sqrt{data_set\ size}$是推荐的取值，但我觉得在数据集规模适宜的情况下最稳妥的方式仍然是通过遍历结合交叉验证找寻到使预测准确率最高的 K 值。

# 四、写在最后

感谢卫金茂老师以及助教学长学姐对本次实验的细致指导和悉心解答。
因初期学习机器学习知识，报告中出现的理解偏差也望指正。

刘嘉洋
NKCS, 1412620
macdowellliu@163.com

2017.3.8