



数据挖掘工具 WEKA教程

广东外语外贸大学
杜剑峰



WEKA教程

1. WEKA简介
2. 数据格式
3. 数据准备
4. 属性选择
5. 可视化分析
6. 分类预测
7. 关联分析
8. 聚类分析
9. 扩展WEKA

课程的总体目标和要求:

- ❖ 熟悉WEKA的基本操作，了解WEKA的各项功能
- ❖ 掌握数据挖掘实验的流程
 - 🔗 准备数据
 - 🔗 选择算法和参数运行
 - 🔗 评估实验结果
- ❖ 了解或掌握在WEKA中加入新算法的方法

1、WEKA简介

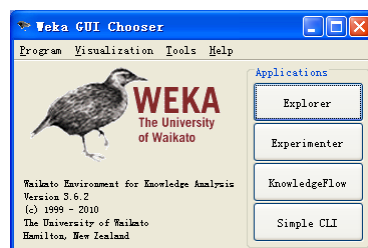
- ❖ WEKA的全名是怀卡托智能分析环境（Waikato Environment for Knowledge Analysis），其源代码可从<http://www.cs.waikato.ac.nz/ml/weka/>得到。同时weka也是新西兰的一种鸟名，而WEKA的主要开发者来自新西兰。



- ❖ 2005年8月，在第11届ACM SIGKDD国际会议上，怀卡托大学的WEKA小组荣获了数据挖掘和知识探索领域的最高服务奖，WEKA系统得到了广泛的认可，被誉为数据挖掘和机器学习历史上的里程碑，是现今最完备的数据挖掘工具之一。WEKA的每月下载次数已超过万次。

1、WEKA简介（续）

- ❖ 作为一个大众化的数据挖掘工作平台，WEKA集成了大量能承担数据挖掘任务的机器学习算法，包括对数据进行预处理、分类、回归、聚类、关联分析以及在新的交互式界面上的可视化等等。通过其接口，可在其基础上实现自己的数据挖掘算法。



WEKA的界面

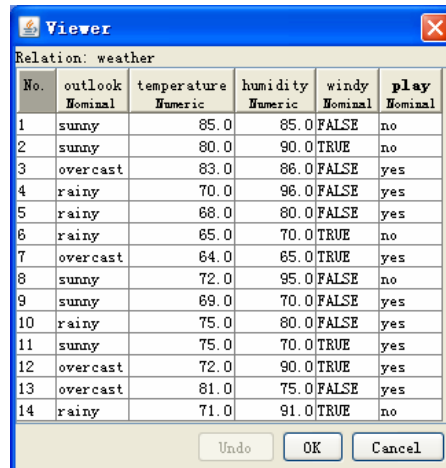
2、数据格式

- ❖ WEKA所用的数据格式（跟Excel一样）

Explorer界面→

Open file...→

Edit...



The screenshot shows the 'Viewer' window in WEKA, displaying a dataset named 'weather'. The window contains a table with 14 instances and 6 attributes. The attributes are 'outlook' (Nominal), 'temperature' (Numeric), 'humidity' (Numeric), 'windy' (Nominal), and 'play' (Nominal). The 'play' attribute is the target class.

No.	outlook Nominal	temperature Numeric	humidity Numeric	windy Nominal	play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

2、数据格式（续）

- ◆ WEKA文件相关术语

表里的一个横行称作一个实例（Instance），相当于统计学中的一个样本，或者数据库中的一条记录。竖行称作一个属性（Attribute），相当于统计学中的一个变量，或者数据库中的一个字段。这样一个表格，或者叫数据集，在WEKA看来，呈现了属性之间的一种关系(Relation)。上图中一共有14个实例，5个属性，关系名称为“weather”。

WEKA存储数据的格式是ARFF（Attribute-Relation File Format）文件，这是一种ASCII文本文件。上图所示的二维表格存储在如下的ARFF文件中。这也就是WEKA自带的“weather.arff”文件，在WEKA安装目录的“data”子目录下可以找到。

```
% ARFF file for the weather data with some numeric features +
% ↓
@relation weather ↓
↓
@attribute outlook {sunny, overcast, rainy} ↓
@attribute temperature real ↓
@attribute humidity real ↓
@attribute windy {TRUE, FALSE} ↓
@attribute play {yes, no} ↓
↓
@data ↓
% ↓
% 14 instances ↓
% ↓
sunny,85,85,FALSE,no ↓
sunny,80,90,TRUE,no ↓
overcast,83,86,FALSE,yes ↓
rainy,70,96,FALSE,yes ↓
rainy,68,80,FALSE,yes ↓
rainy,65,70,TRUE,no ↓
overcast,64,65,TRUE,yes ↓
sunny,72,95,FALSE,no ↓
sunny,69,70,FALSE,yes ↓
rainy,75,80,FALSE,yes ↓
sunny,75,70,TRUE,yes ↓
```

2、数据格式（续）

❖ 文件内容说明

识别ARFF文件的重要依据是分行，因此不能在这种文件里随意的断行。空行（或全是空格的行）将被忽略。以“%”开始的行是注释，WEKA将忽略这些行。如果你看到的“weather.arff”文件多了或少了些“%”开始的行，是没有影响的。

除去注释后，整个ARFF文件可以分为两个部分。

- ❖ 第一部分给出了头信息（Head information），包括了对关系的声明和对属性的声明。
- ❖ 第二部分给出了数据信息（Data information），即数据集中给出的数据。从“@data”标记开始，后面的就是数据信息了。

2、数据格式（续）

❖ 关系声明

关系名称在ARFF文件的第一个有效行来定义，格式为
@relation <relation-name>

<relation-name>是一个字符串。如果这个字符串包含空格，它必须加上引号（指英文标点的单引号或双引号）。

2、数据格式（续）

❖ 属性声明

属性声明用一系列以“**@attribute**”开头的语句表示。数据集中的每一个属性都有它对应的“**@attribute**”语句，来定义它的属性名称和数据类型。

这些声明语句的顺序很重要。首先它表明了该项属性在数据部分的位置。例如，“**humidity**”是第三个被声明的属性，这说明数据部分那些被逗号分开的列中，第三列数据 **85 90 86 96 ...** 是相应的“**humidity**”值。其次，最后一个声明的属性被称作**class**属性，在分类或回归任务中，它是默认的目标变量。

属性声明的格式为

@attribute <attribute-name> <datatype>

其中**<attribute-name>**是必须以字母开头的字符串。和关系名称一样，如果这个字符串包含空格，它必须加上引号。

2、数据格式（续）

- ❖ WEKA支持的<datatype>有四种

↪ numeric	数值型
↪ <nominal-specification>	标称（nominal）型
↪ string	字符串型
↪ date [<date-format>]	日期和时间型

其中<nominal-specification> 和<date-format> 将在下面说明。还可以使用两个类型“integer”和“real”，但是WEKA把它们都当作“numeric”看待。注意“integer”，“real”，“numeric”，“date”，“string”这些关键字是区分大小写的，而“relation”、“attribute”和“data”则不区分。

2、数据格式（续）

- ❖ 数值属性
数值型属性可以是整数或者实数，但WEKA把它们都当作实数看待。
- ❖ 标称属性
标称属性由<nominal-specification>列出一系列可能的类别名称并放在花括号中：{<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}。数据集中该属性的值只能是其中一种类别。
例如如下的属性声明说明“outlook”属性有三种类别：“sunny”，“overcast”和“rainy”。而数据集中每个实例对应的“outlook”值必是这三者之一。
@attribute outlook {sunny, overcast, rainy}
如果类别名称带有空格，仍需要将之放入引号中。

2、数据格式（续）

- ◆ 字符串属性
字符串属性中可以包含任意的文本。这种类型的属性在文本挖掘中非常有用。
示例：
`@ATTRIBUTE LCC string`
- ◆ 日期和时间属性
日期和时间属性统一用“date”类型表示，它的格式是
`@attribute <name> date [<date-format>]`
其中<name>是这个属性的名称，<date-format>是一个字符串，来规定怎样解析和显示日期或时间的格式，默认的字符串是ISO-8601所给的日期时间组合格式“`yyyy-MM-ddTHH:mm:ss`”。数据信息部分表达日期的字符串必须符合声明中规定的格式要求（下文有例子）。

2、数据格式（续）

- ◆ 数据信息
数据信息中“`@data`”标记独占一行，剩下的是各个实例的数据。

每个实例占一行。实例的各属性值用逗号“,”隔开。如果某个属性的值是缺失值（missing value），用问号“?”表示，且这个问号不能省略。例如：
`@data`
`sunny,85,85,FALSE,no`
`?,78,90,?,yes`

2、数据格式（续）

- ◆ 字符串属性和标称属性的值是区分大小写的。若值中含有空格，必须被引号括起来。例如：

```
@relation LCCvsLCSH
@attribute LCC string
@attribute LCSH string
@data
AG5, 'Encyclopedias and dictionaries.;Twentieth century.'
AS262, 'Science -- Soviet Union -- History.'
```

2、数据格式（续）

- ◆ 日期属性的值必须与属性声明中给定的相一致。例如：

```
@RELATION Timestamps
@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss"
@DATA
"2001-04-03 12:12:12"
"2001-05-03 12:59:55"
```


3、数据准备

❖ 数据文件格式转换

使用WEKA作数据挖掘，面临的第一个问题往往是我们的数据不是ARFF格式的。幸好，WEKA还提供了对CSV文件的支持，而这种格式是被很多其他软件，比如Excel，所支持的。现在我们打开“bank-data.csv”。

利用WEKA可以将CSV文件格式转化成ARFF文件格式。ARFF格式是WEKA支持得最好的文件格式。

此外，WEKA还提供了通过JDBC访问数据库的功能。

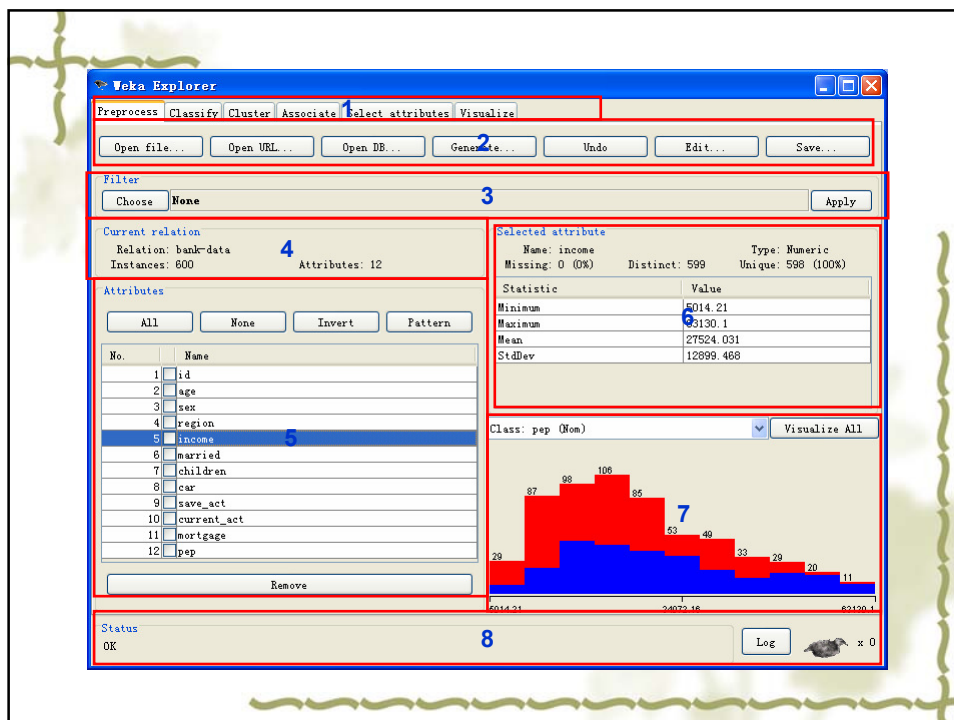
❖ “Explorer”界面

“Explorer”提供了很多功能，是WEKA使用最多的模块。现在我们先来熟悉它的界面，然后利用它对数据进行预处理。

3、数据准备（续）

❖ bank-data数据各属性的含义如下：

id:	a unique identification number
age:	age of customer in years (numeric)
sex:	MALE / FEMALE
region:	inner_city/rural/suburban/town
income:	income of customer (numeric)
married:	is the customer married (YES/NO)
children:	number of children (numeric)
car:	does the customer own a car (YES/NO)
save_act:	does the customer have a saving account (YES/NO)
current_act:	does the customer have a current account (YES/NO)
mortgage:	does the customer have a mortgage (YES/NO)
pep:	did the customer buy a PEP (Personal Equity Plan, 个人参股计划) after the last mailing (YES/NO)



3、数据准备（续）

- ❖ 上图显示的是“Explorer”打开“bank-data.csv”的情况。我们根据不同的功能把这个界面分成8个区域。
- 1. 区域1的几个选项卡是用来切换不同的挖掘任务面板。
- 2. 区域2是一些常用按钮。包括打开数据，保存及编辑功能。我们可以在这里把“bank-data.csv”另存为“bank-data.arff”。
- 3. 在区域3中“Choose”某个“Filter”，可以实现筛选数据或者对数据进行某种变换。数据预处理主要就利用它来实现。
- 4. 区域4展示了数据集的一些基本情况。
- 5. 区域5中列出了数据集的所有属性。勾选一些属性并“Remove”就可以删除它们，删除后还可以利用区域2的“Undo”按钮找回。区域5上方的一排按钮是用来快速勾选的。在区域5中选中某个属性，则区域6中有关于这个属性的摘要。注意对于数值属性和标称属性，摘要的方式是不一样的。图中显示的是对数值属性“income”的摘要。

3、数据准备（续）

6. 区域7是区域5中选中属性的直方图。若数据集的某个属性是目标变量，直方图中的每个长方形就会按照该变量的比例分成不同颜色的段。默认地，分类或回归任务的默认目标变量是数据集的最后一个属性（这里的“**pep**”正好是）。要想换个分段的依据，即目标变量，在区域7上方的下拉框中选个不同的分类属性就可以了。下拉框里选上“**No Class**”或者一个数值属性会变成黑白的直方图。
7. 区域8是状态栏，可以查看**Log**以判断是否有错。右边的**weka**鸟在动的话说明**WEKA**正在执行挖掘任务。右键点击状态栏还可以执行**JAVA**内存的垃圾回收。

3、数据准备（预处理1）

❖ 删除无用属性

通常对于数据挖掘任务来说，ID这样的信息是无用的，我们将之删除。在区域5勾选属性“id”，并点击“**Remove**”。将新的数据集保存为“**bank-data.arff**”，重新打开。此外，我们可以通过名为“**RemoveType**”的Filter删除某一类型的属性。

❖ 离散化

我们知道，有些算法(如关联分析)，只能处理所有的属性都是标称型的情况。这时候我们就需要对数值型的属性进行离散化。在这个数据集中有3个变量是数值型的，分别是“age”，“income”和“children”。其中“children”只有4个取值：0，1，2，3。这时我们可以通过名为“**NumericToNominal**”的Filter将children的类型变成Nominal。

3、数据准备（预处理2）

- ❖ 离散化（续）
- ◆ “age”和“income”的离散化可借助WEKA中名为“Discretize”的Filter来完成。
现在“Choose”旁边的文本框应该显示“Discretize -B 10 -M 0.1 -R first-last”。点击这个文本框会弹出新窗口以修改离散化的参数。
我们不打算对所有的属性离散化，只是针对对第1个和第4个属性（见区域5属性名左边的数字），故把attributeIndices右边改成“1,4”。计划把这两个属性都分成3段，于是把“bins”改成“3”。其它框里不用更改。点“OK”回到“Explorer”，可以看到“age”和“income”已经被离散化成分类型的属性。若想放弃离散化可以点区域2的“Undo”。
- ◆ 经过上述操作得到的数据集我们保存为 bank-data-final.arff。

3、数据准备（预处理3）

- ❖ 属性类型转换
NominalToBinary过滤器将所有nominal类型的属性转为binary(0,1二值)属性，一个可取k个值的nominal类型的属性转为k个二值属性，这样可将数据中所有属性转为数值(numeric)属性。以下是weather.arff转换后的结果。

No.	outlook=sunny Numeric	outlook=overcast Numeric	outlook=rainy Numeric	temperature Numeric	humidity Numeric	windy Numeric	play Nominal
1	1.0	0.0	0.0	85.0	85.0	1.0	no
2	1.0	0.0	0.0	80.0	90.0	0.0	no
3	0.0	1.0	0.0	83.0	86.0	1.0	yes
4	0.0	0.0	1.0	70.0	96.0	1.0	yes
5	0.0	0.0	1.0	68.0	80.0	1.0	yes

3、数据准备（预处理4）

- ❖ 增加一个表达式属性
- ◆ **AddExpression**: An instance filter that creates a new attribute by applying a mathematical expression to existing attributes. The expression can contain attribute references and numeric constants. Supported operators are : +, -, *, /, ^, log, abs, cos, exp, sqrt, floor, ceil, rint, tan, sin, (,)
- ◆ Attributes are specified by prefixing with 'a', eg. a7 is attribute number 7 (starting from 1).
Example expression : $a1^2 * a5 / \log(a7 * 4.0)$.
以下命令在weather.arff中增加了一个temp/hum属性，其值为第二个属性（temperature）除以第三个属性（humidity）的值。
AddExpression -E a2/a3 -N temp/hum

3、数据准备（预处理5）

- ❖ 采样
- ◆ 使用weka.filters.supervised.instance.Resample对整个数据集进行分层的采样(stratified subsample, 采样所得数据仍保持原来的类分布)。以下Filter命令从soybean.arff中采样了5%的数据。
Resample -B 0.0 -S 1 -Z 5.0
- ◆ 使用weka.filters.unsupervised.instance.Resample进行不分层的采样，即与类信息无关。以下Filter命令从soybean.arff中采样了5%的数据。
Resample -S 1 -Z 5.0

4、属性选择

- ❖ 两种属性子集选择模式
 - ⌚ 属性子集评估器 + 搜索方法
 - ⌚ 单一属性评估器 + 排序方法

4.1 属性选择模式1

- | | |
|--|---|
| <ul style="list-style-type: none">❖ 属性子集评估器<ul style="list-style-type: none">⌚ CfsSubsetEval: 综合考虑单一属性的预测值和属性间的重复度⌚ ClassifierSubsetEval: 用分类器评估属性集⌚ ConsistencySubsetEval: 将训练数据集映射到属性集上来检测类值的一致性⌚ WrapperSubsetEval: 使用分类器和交叉验证（包装方法） | <ul style="list-style-type: none">❖ 搜索方法<ul style="list-style-type: none">⌚ BestFirst: 回溯的贪婪搜索⌚ ExhaustiveSearch: 穷举搜索（穷举方法）⌚ GeneticSearch: 使用遗传算法搜索⌚ GreedyStepwise: 不回溯的贪婪搜索⌚ RandomSearch: 随机搜索⌚ RankSearch: 排列属性并使用属性子集评估器将有潜力的属性进行排序 |
|--|---|

4.2、属性选择模式2

- ❖ 单一属性评估器
 - 🔗 **ChiSquaredAttributeEval**: 以基于类的 χ^2 为依据的属性评估
 - 🔗 **GainRatioAttributeEval**: 以增益率为依据的属性评估
 - 🔗 **InfoGainAttributeEval**: 以信息增益为依据的属性评估
 - 🔗 **OneRAttributeEval**: 以OneR的方法论来评估属性
 - 🔗 **PrincipleComponents**: 进行主成分的分析 and 转换
 - 🔗 **ReliefAttributeEval**: 基于实例的属性评估器
 - 🔗 **SymmetricalUncertAttributeEval**: 以对称不确定性为依据的属性评估
- ❖ 排序方法
 - 🔗 **Ranker**: 按照属性的评估对它们进行排序

5、可视化分析

- ❖ 二维散列图
 - 🔗 选择类标
 - ❖ 标称类标: 数据点的颜色是离散的
 - ❖ 数值类标: 数据点的颜色用色谱（蓝色到橙色）表示
 - 🔗 改变点阵的大小和点的大小
 - 🔗 改变抖动度，使互相重叠的点分开
 - 🔗 选择属性子集和采样
 - 🔗 注意：必须点击**Update**按钮上述改动才能生效

6、分类预测

- WEKA把分类(Classification)和回归(Regression)都放在“Classify”选项卡中。

在这两个任务中，都有一个目标属性（即输出变量或类标）。我们希望根据一个样本(WEKA中称作实例)的一组特征（输入变量），对目标进行预测。为了实现这一目的，我们需要有一个训练数据集，这个数据集中每个实例的输入和输出都是已知的。观察训练集中的实例，可以建立起预测的模型。有了这个模型，我们就可以对新的未知实例进行预测了。衡量模型的好坏主要在于预测的准确率。

选择分类算法

❖ WEKA中的典型分类算法

☞ Bayes: 贝叶斯分类器

❖ BayesNet: 贝叶斯信念网络

❖ NaïveBayes: 朴素贝叶斯网络

☞ Functions: 人工神经网络和支持向量机

❖ MultilayerPerceptron: 多层前馈人工神经网络

❖ SMO: 支持向量机（采用顺序最优化学习方法）

☞ Lazy: 基于实例的分类器

❖ IB1: 1-最近邻分类器

❖ IBk: k-最近邻分类器

选择分类算法

☞ Meta: 组合方法

- ❖ AdaBoostM1: AdaBoost M1方法
- ❖ Bagging: 袋装方法

☞ Rules: 基于规则的分类器

- ❖ JRip: 直接方法—Ripper算法
- ❖ Part: 间接方法—从J48产生的决策树抽取规则（不是C4.5规则算法）

☞ Trees: 决策树分类器

- ❖ Id3: ID3决策树学习算法（不支持连续属性）
- ❖ J48: C4.5决策树学习算法（第8版本）
- ❖ RandomForest: 基于决策树的组合方法

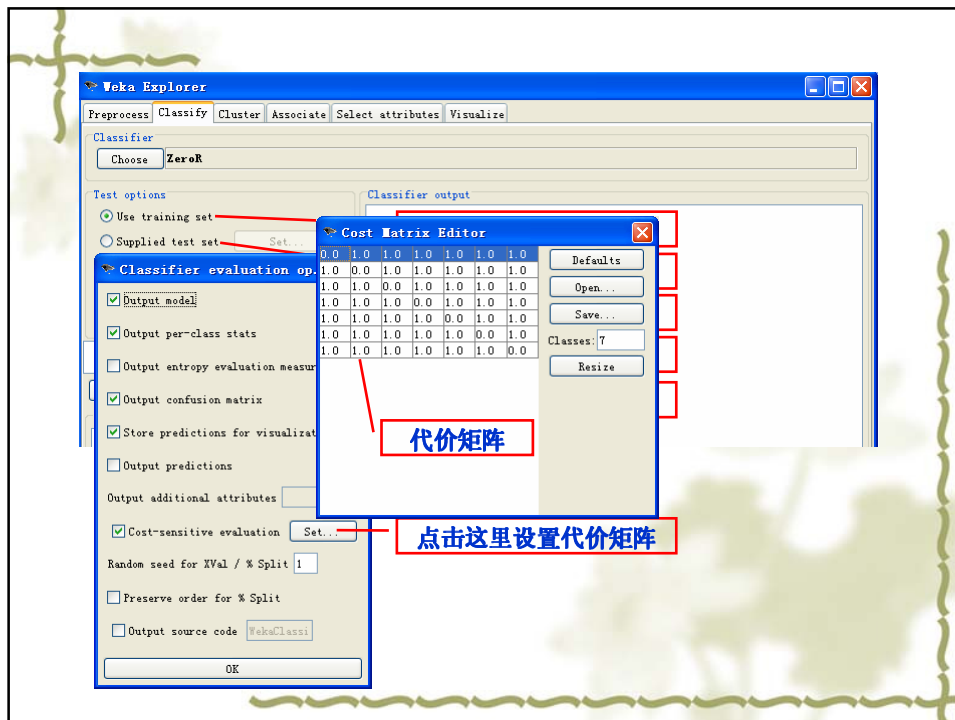
选择模型评估方法

❖ 四种方法

- ☞ 使用训练集作为测试集
- ☞ 使用外部的测试集
- ☞ 交叉验证
 - ❖ 设置折数
- ☞ 保持方法
 - ❖ 设置训练实例的百分比

❖ 其他设置

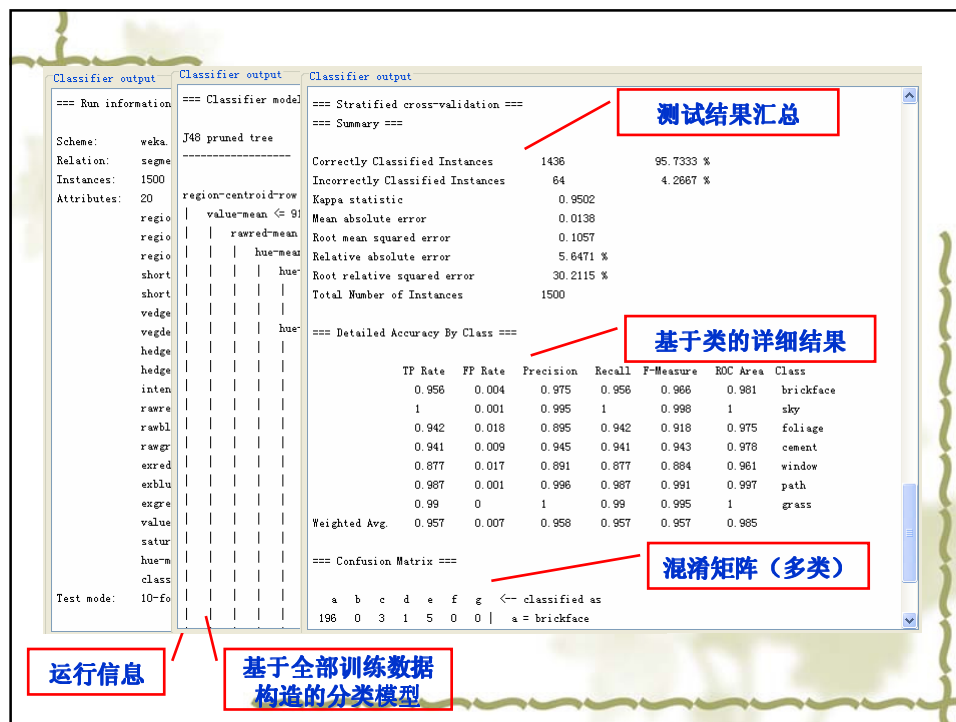
- ☞ 设置代价矩阵



文字结果分析

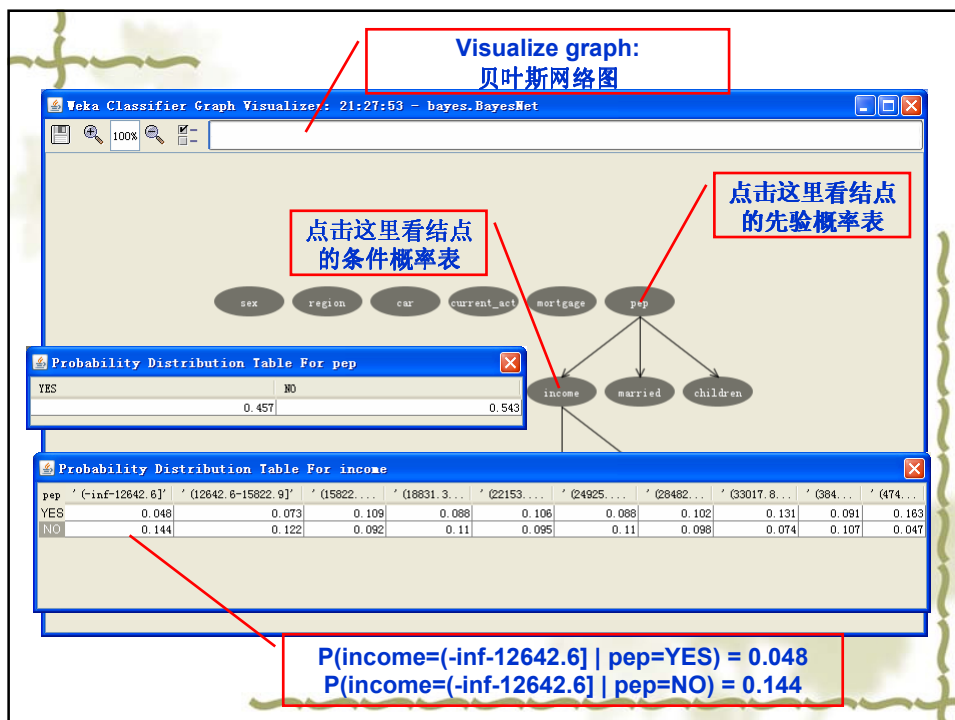
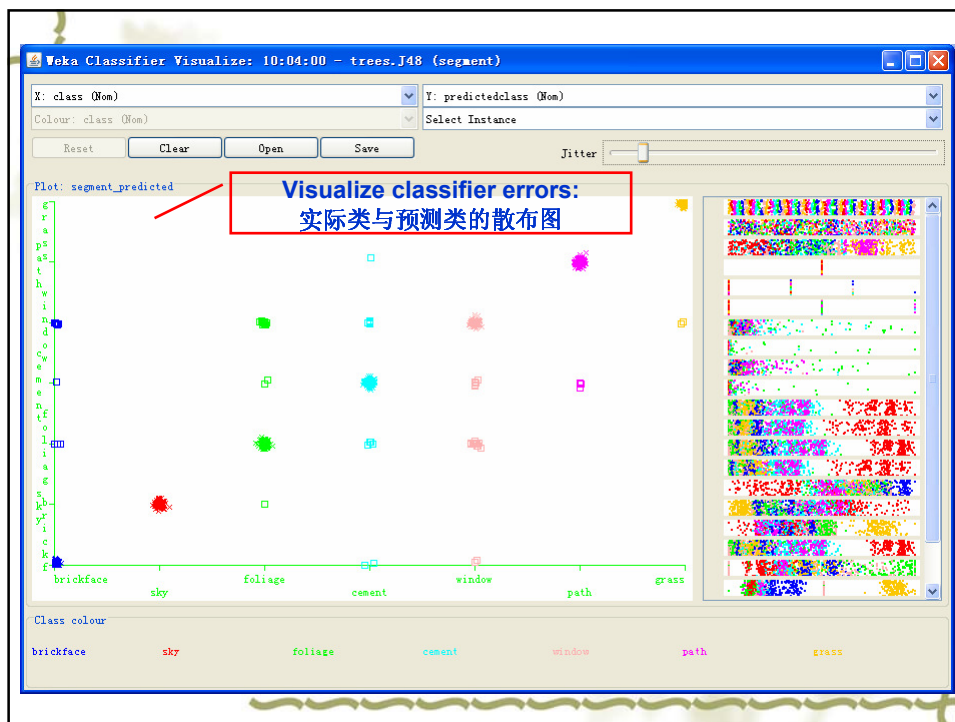
窗口显示的文字结果信息：

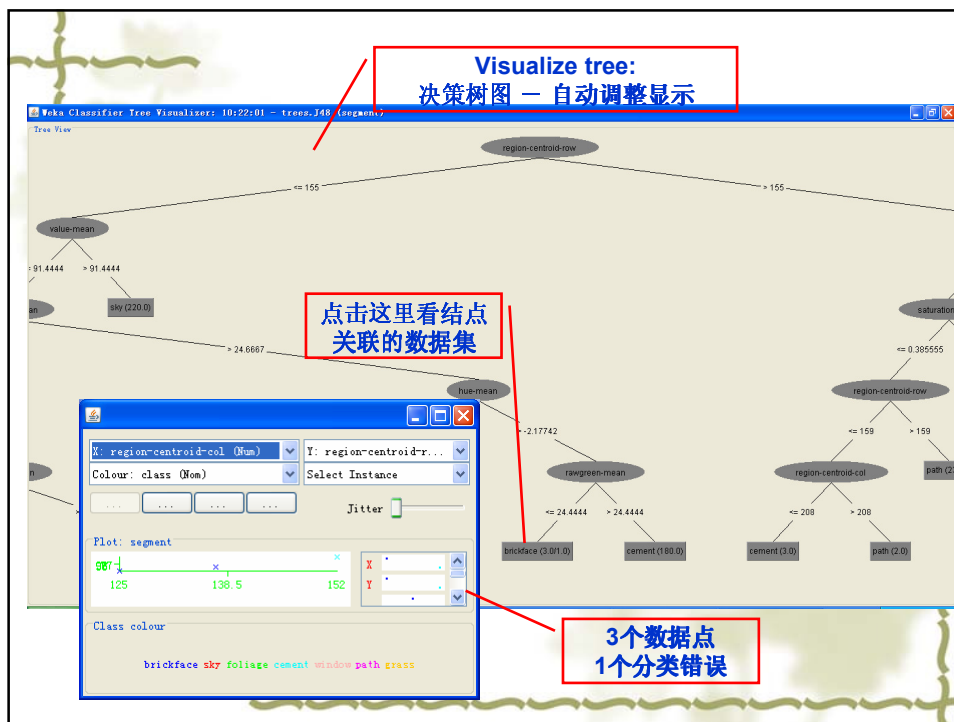
- ❖ 运行信息
- ❖ 使用全部训练数据构造的分类模型
- ❖ 针对测试集的测试结果汇总
 - ↪ k -折交叉验证的结果是 k 次实验的汇总
 - 即 $TP=TP_1+\dots+TP_k$, $FN=FN_1+\dots+FN_k$,
 $FP=FP_1+\dots+FP_k$, $TN=TN_1+\dots+TN_k$
- ❖ 基于类的详细结果
 - ↪ 加权平均的系数是类大小的百分比
- ❖ 混淆矩阵



图形结果分析

- ❖ 可视化分类错误
 - ☞ 实际类与预测类的散布图
- ❖ 可视化模型
 - ☞ 可视化图：贝叶斯网络
 - ❖ 查看条件概率表
 - ☞ 可视化树：决策树
 - ❖ 居中显示
 - ❖ 屏幕大小显示
 - ❖ 自动调整显示
 - ❖ 查看结点关联的训练集

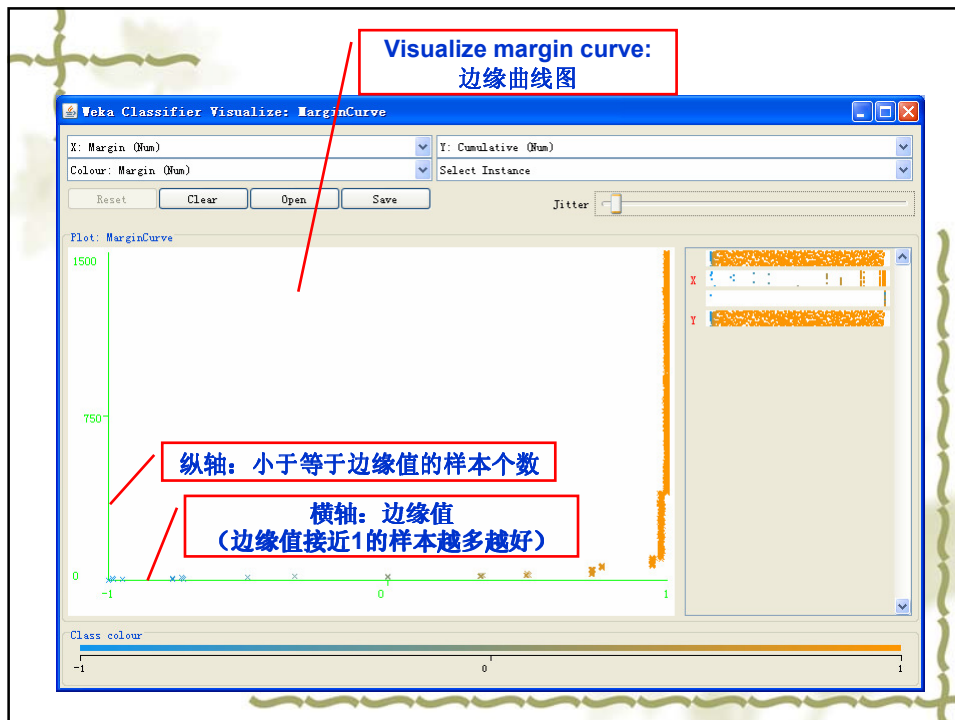




图形结果分析...

❖ 可视化边缘曲线 (margin curve)

- ☞ 显示预测边缘，即实际类的预测概率与其他类的最大预测概率的差别
- ☞ 对于每个测试样本，从小到大显示预测边缘
- ☞ 四个变量
 - ❖ **Margin**: 预测边缘的值
 - ❖ **Instance_number**: 测试样本的序号
 - ❖ **Current**: 具有当前预测边缘值的样本个数
 - ❖ **Cumulative**: 小于或等于预测边缘值的样本个数（与 Instance_number 一致）



图形结果分析...

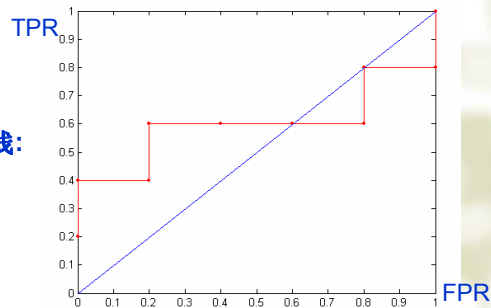
- ❖ 可视化阈值曲线（基于类）
 - 🔗 阈值是将检验实例归为当前类的最小概率，使用点的颜色表示阈值
 - 🔗 曲线上的每个点通过改变阈值的大小生成
 - 🔗 可以进行ROC分析
 - ❖ X轴选假正率（false positive）
 - ❖ Y轴选真正率（true positive）
 - 🔗 ROC曲线 — 接受者操作特征(Receiver Operating Characteristic)曲线: 描绘 真正率TPR (纵轴) 随着 假正率FPR (横轴) 变化的趋势
 - 🔗 AUC: ROC曲线下方的区域面积（理想情况AUC=1）

如何构造ROC曲线

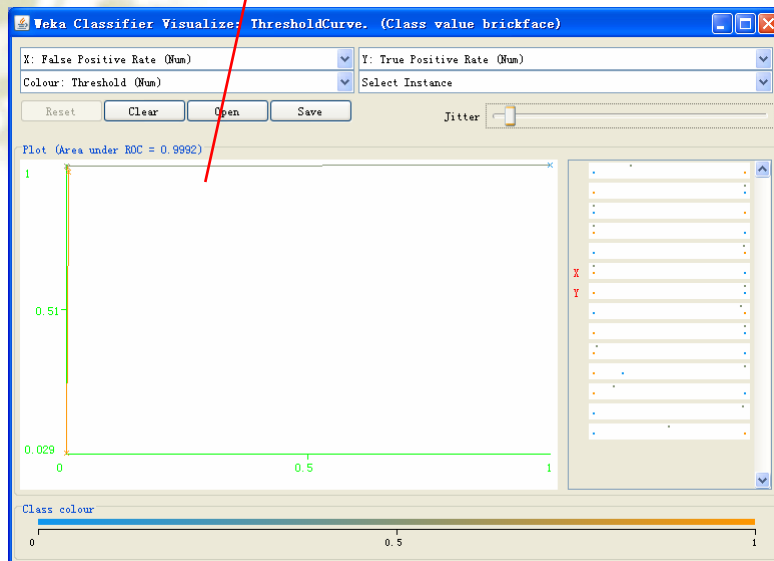
Class	+	-	+	-	-	-	+	-	+	+	
测试样本概率阈值 >=	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

测试样本属于+类的概率

ROC曲线:



Visualize threshold curve:
基于某个类的ROC曲线

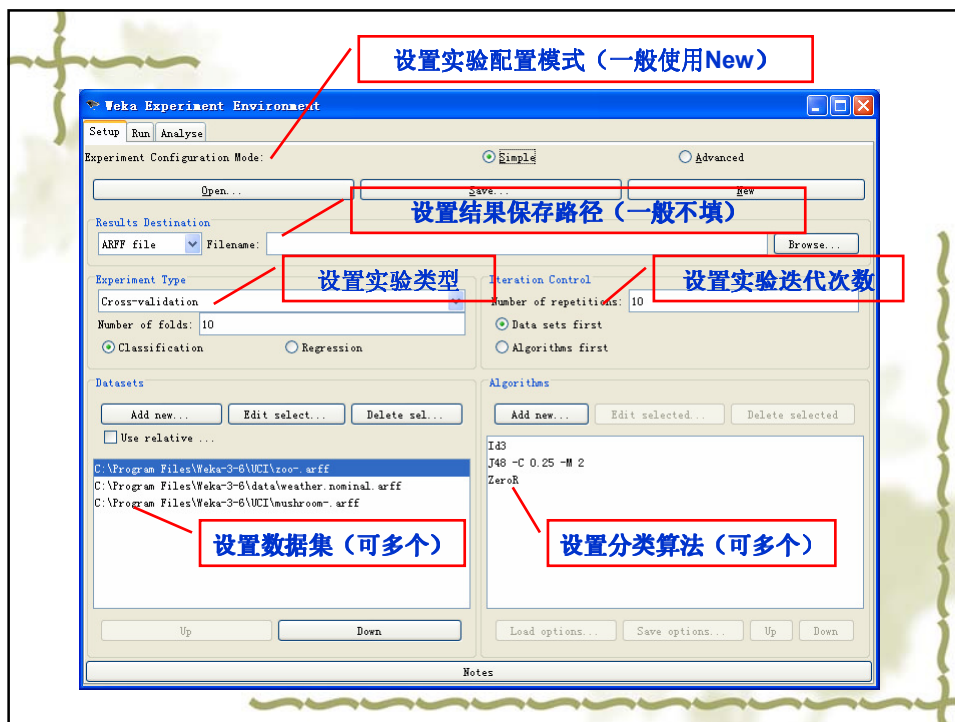


实验者界面

- ❖ 优点
 - 🔗 同时对多个数据集和多个分类算法工作
 - 🔗 可以比较多个分类算法的性能
- ❖ 缺点
 - 🔗 不能使用数据预处理工具
 - 🔗 不能选择类标，只能将输入数据集的最后一个属性作为类标
- ❖ 三个页面
 - 🔗 设置页面 (Setup) — 设置实验参数
 - 🔗 运行页面 (Run) — 启动实验，监视实验过程
 - 🔗 分析页面 (Analyze) — 分析实验结果

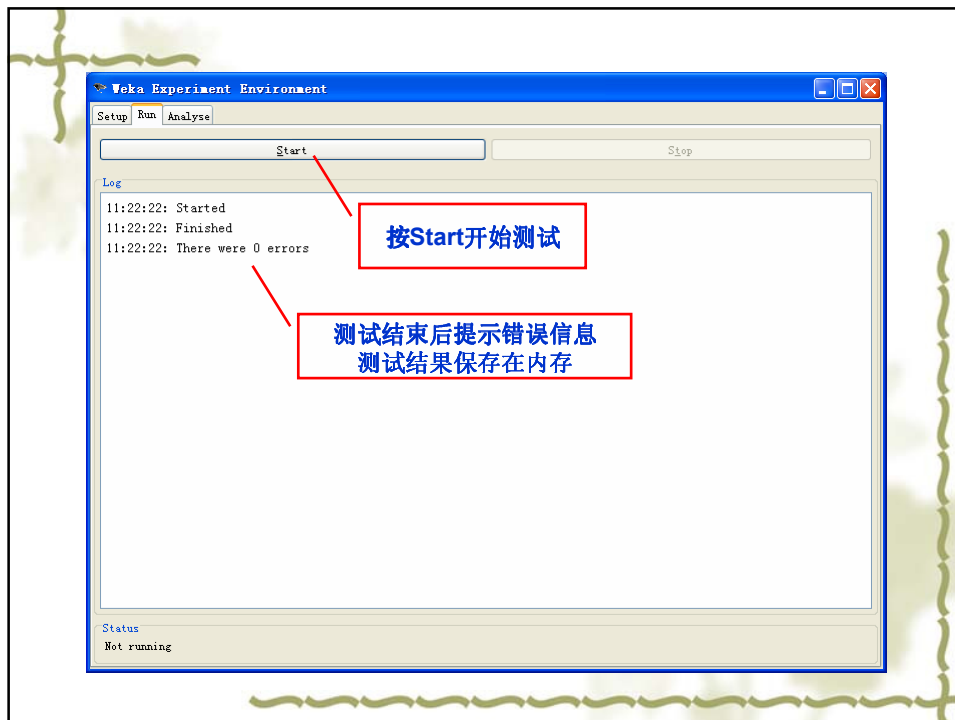
设置页面

- ❖ 设置实验配置模式 (Experiment Configuration Mode)
- ❖ 设置结果保存路径 (Choose Destination)
- ❖ 设置实验类型 (Experiment Type)
 - 🔗 交叉验证
 - 🔗 保持方法（随机化记录次序）
 - 🔗 保持方法（维持原有记录次序）
- ❖ 迭代控制 (Iteration Control)
 - 🔗 设置实验迭代次数 —— 减少数据抽样随机性的影响
- ❖ 数据集 (Datasets)
 - 🔗 增加数据集，类标是数据集最后一个属性
- ❖ 分类算法 (Algorithms)
 - 🔗 增加算法
 - 🔗 设置算法参数



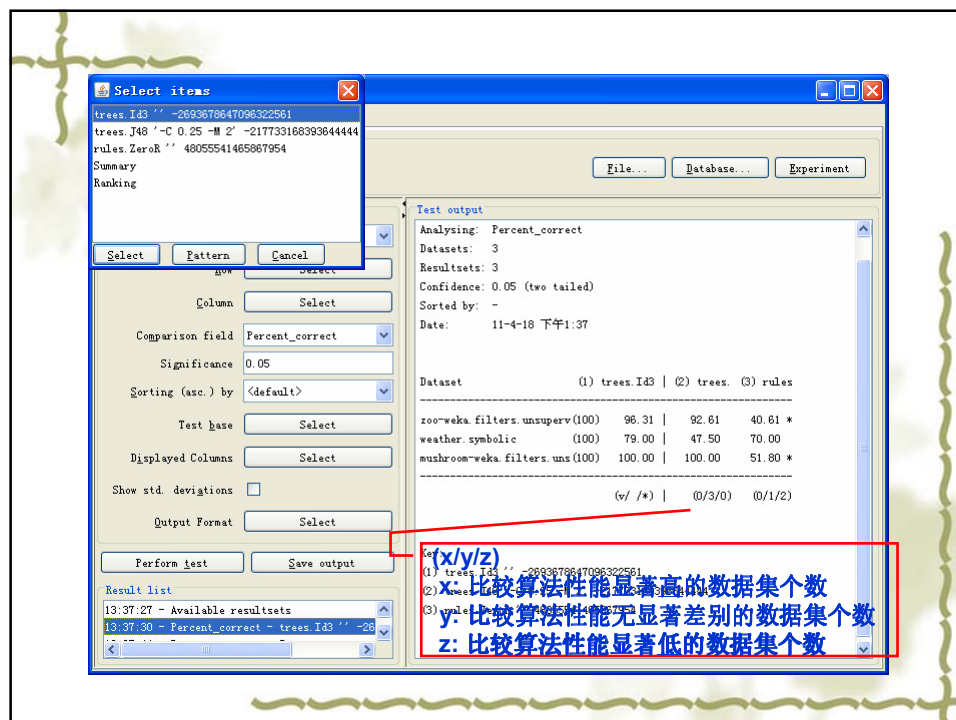
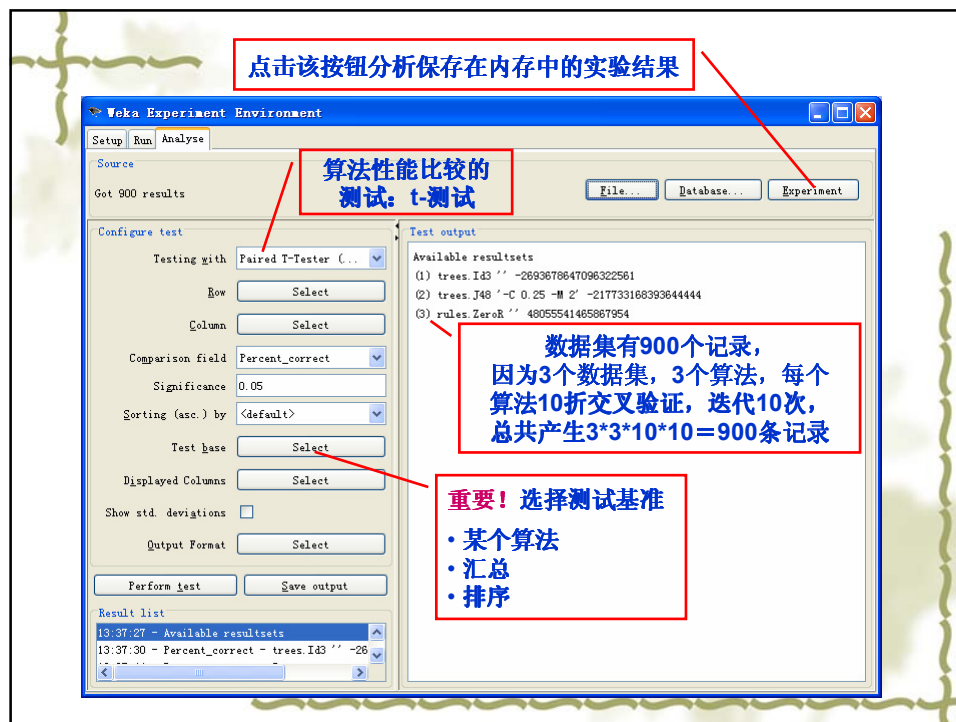
运行页面

- ❖ 点击运行，报告运行情况
- ❖ 运行后生成一个数据集
 - ☞ 一个记录对应一个数据集和一个分类算法的一次实验，字段包括算法、数据集和不同的性能度量
 - ☞ 该数据集保存在内存中
 - ☞ 分析仅限于算法性能比较的显著性检验
 - ☞ 没有可视化分析功能



分析页面

- ❖ 实验结果数据源 (Source)
- ❖ 配置测试 (Configure test)
 - 🔗 选择行和列，行缺省是数据集，列缺省是Scheme, Scheme_options和Scheme_version_ID
 - 🔗 测试基准 (Test base)
 - ❖ 某个算法
 - ❖ 汇总 (summary)
 - ❖ 排序 (ranking)
- ❖ 结果列表 (Result list)
- ❖ 测试输出 (Test output)



实验内容

- ❖ 分组对UCI数据集进行实验
 - 🌀 15~16个组
 - 🌀 每组选择一个数据集分析
- ❖ 实验内容
 1. 使用一个UCI数据集，评估至少三个分类算法的性能
 - ❖ 解释哪个算法的性能最好，是否显著地好
 2. 分析性能最好的算法的实验结果
 - ❖ 解释文字部分的性能评估结果
 - ❖ 解释图形部分的性能评估结果
 - ❖ 解释分类模型

知识流界面

- ❖ 功能：将WEKA组件在设计画布上相互连接以形成可进行动态数据处理分析的知识流
- ❖ 两种数据处理模式
 1. 批量处理
 - 🌀 探索者界面支持的测试模式，在知识流界面都支持
 2. 增量处理
 - 探索者界面不支持增量处理数据
 - 目前WEKA实现了下面这些可增量学习的分类器：
AODE、IB1、IBk、KStar、
NaiveBayesMultinomialUpdateable、
NaiveBayesUpdateable、NNge、Winnow、
RacedIncrementalLogitBoost、LWL

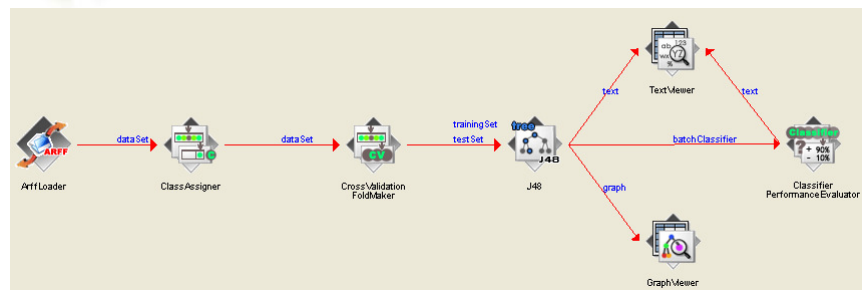
批量处理模式（交叉验证）

❖ 交叉验证J48显示

- ☞ Datasources – ArffLoader
- ☞ Evaluation – ClassAssigner
- ☞ Evaluation – CrossValidationFoldMaker
- ☞ Classifiers – J48
- ☞ Evaluation – ClassifierPerformanceEvaluator
- ☞ Visualization – TextViewer
- ☞ Visualization – GraphViewer

批量处理模式（交叉验证）

❖ 交叉验证J48显示



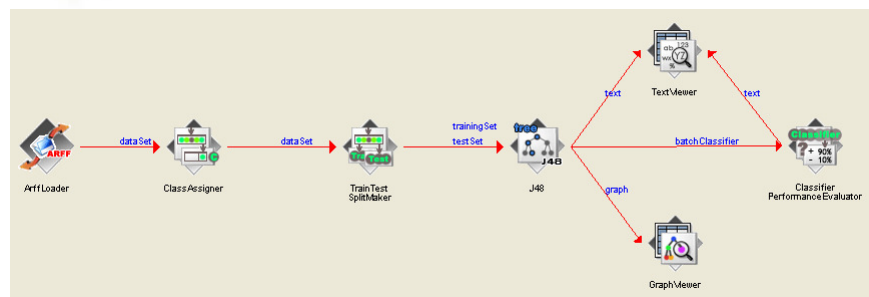
批量处理模式（保持方法）

❖ 保持方法J48显示

- 🔗 Datasources – ArffLoader
- 🔗 Evaluation – ClassAssigner
- 🔗 Evaluation – TrainTestSplitMaker
- 🔗 Classifiers – J48
- 🔗 Evaluation – ClassifierPerformanceEvaluator
- 🔗 Visualization – TextViewer
- 🔗 Visualization – GraphViewer

批量处理模式（保持方法）

❖ 保持方法J48显示

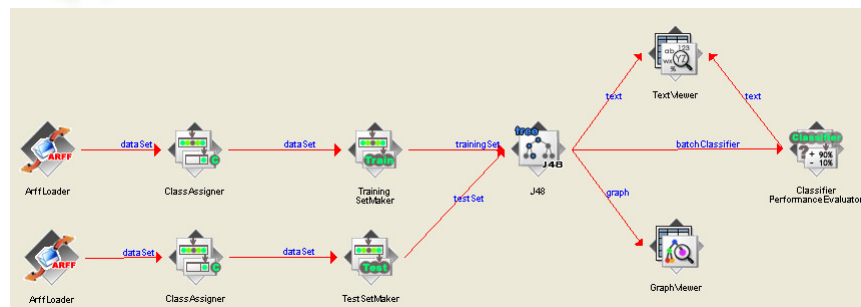


批量处理模式（外部测试集）

- ❖ 使用外部测试集J48显示
 - 🔗 Datasources – ArffLoader × 2
 - 🔗 Evaluation – ClassAssigner × 2
 - 🔗 Evaluation – TrainingSetMaker
 - 🔗 Evaluation – TestSetMaker
 - 🔗 Classifiers – J48
 - 🔗 Evaluation – ClassifierPerformanceEvaluator
 - 🔗 Visualization – TextViewer
 - 🔗 Visualization – GraphViewer

批量处理模式（外部测试集）

- ❖ 使用外部测试集J48显示



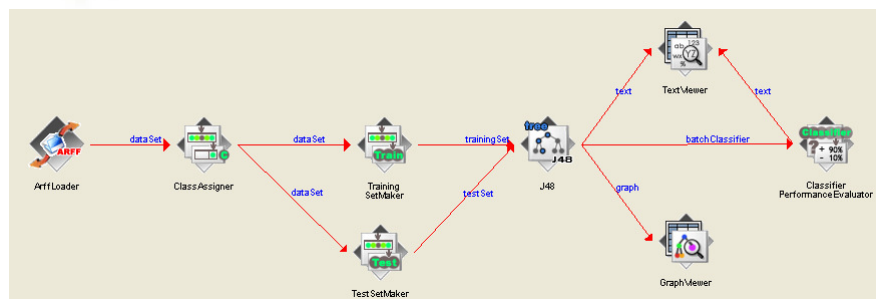
批量处理模式（训练集用于测试）

❖ 训练集用于测试的J48显示

- 🔗 Datasources – ArffLoader
- 🔗 Evaluation – ClassAssigner
- 🔗 Evaluation – TrainingSetMaker
- 🔗 Evaluation – TestSetMaker
- 🔗 Classifiers – J48
- 🔗 Evaluation – ClassifierPerformanceEvaluator
- 🔗 Visualization – TextViewer
- 🔗 Visualization – GraphViewer

批量处理模式（训练集用于测试）

❖ 训练集用于测试的J48显示

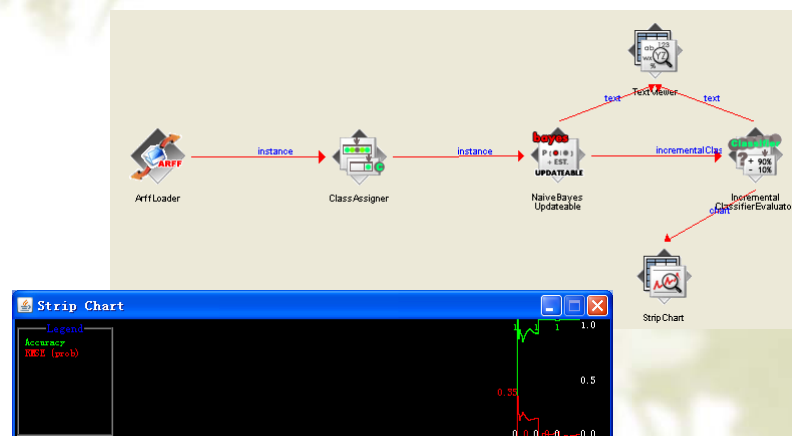


增量处理模式

- ❖ 增量学习 NaiveBayesUpdateable 显示
 - 🔗 Datasources – ArffLoader
 - 🔗 Evaluation – ClassAssigner
 - 🔗 Classifiers – NaiveBayesUpdateable
 - 🔗 Evaluation – IncrementalClassifierEvaluator
 - 🔗 Visualization – TextViewer
 - 🔗 Visualization – StripChart
 - ❖ Accuracy – 准确率
 - ❖ RMSE – 均方根误差 (root-mean-square error)

增量处理模式

- ❖ 增量学习 NaiveBayesUpdateable 显示



7、关联分析

- ❖ 对于关联规则 $L \rightarrow R$ ，由支持度决定规则的统计显著性，并由四种不同的因素之一决定规则的优先度。
- ❖ 支持度（support）—— 同时观察到前件和后件的概率
 $\text{support} = \text{Pr}(L, R)$
- ❖ 置信度（confidence）—— 出现前件时同时出现后件的概率
 $\text{confidence} = \text{Pr}(L, R) / \text{Pr}(L)$
- ❖ 提升度（lift）—— 置信度与后件支持度的比率
 $\text{lift} = \text{Pr}(L, R) / (\text{Pr}(L)\text{Pr}(R))$
- ❖ 平衡度（leverage）—— 在前件和后件统计独立的假设下，被前件和后件同时涵盖的超出期望值的那部分实例的比例
 $\text{leverage} = \text{Pr}(L, R) - \text{Pr}(L)\text{Pr}(R)$
- ❖ 可信度（conviction）—— 也用来衡量前件和后件的独立性
 $\text{conviction} = \text{Pr}(L)\text{Pr}(\text{not } R) / \text{Pr}(L, R)$

基本的关联分析操作

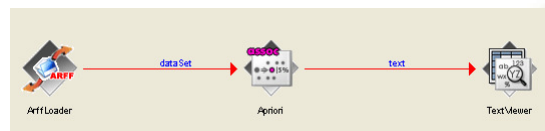
- ❖ Soybean数据的关联分析
用“Explorer”打开“soybean.arff”后，切换到“Associate”选项卡。默认关联规则分析是用Apriori算法。
点“Choose”右边的文本框修改默认的参数，弹出的窗口中点“More”可以看到各参数的说明。
点击“Start”按钮开始关联分析。
- ❖ WEKA中关联分析的过程
假设最小置信度设定为0.9，支持度上限1，支持度下限0.1：
从数据项的支持度上限100%-5%开始，逐步递减5%，直到至少有满足置信度条件（即 $\geq 90\%$ ）的10条规则，或者支持度达到了10%的下限。

参数设置练习

- ❖ 数据集为“weather.nominal.arff”
- ❖ 任务一：挖掘支持度在10%到100%之间，并且提升度超过1.5且提升度排在前100位的关联规则
 - ⌚ “lowerBoundMinSupport”和“upperBoundMinSupport”分别设为0.1和1
 - ⌚ “metricType”设为lift
 - ⌚ “minMetric”设为1.5
 - ⌚ “numRules”设为100
- ❖ 任务二：挖掘支持度在10%到100%之间，并且置信度超过0.8且置信度排在前100位的分类关联规则
 - ⌚ “car”设为True
 - ⌚ “metricType”设为confidence (只能选confidence!)
 - ⌚ “minMetric”设为0.8
 - ⌚ “numRules”设为100

知识流界面运行

- ❖ 挖掘支持度在10%到100%之间，并且置信度超过0.8且置信度排在前100位的分类关联规则
 - ⌚ 数据集为“weather.nominal.arff”
 - ⌚ “car”设为True
 - ⌚ “metricType”设为confidence (只能选confidence!)
 - ⌚ “minMetric”设为0.8
 - ⌚ “numRules”设为100



8、聚类分析

- ❖ 聚类分析是把对象分配给各个簇，使同簇中的对象相似，而不同簇间的对象相异。
- ❖ WEKA在“Explorer”界面的“Cluster”提供聚类分析工具，主要算法包括：
 - 🔗 **SimpleKMeans** — 支持分类属性的K均值算法
 - 🔗 **DBScan** — 支持分类属性的DBSCAN算法
 - 🔗 **EM** — 基于混合模型的聚类算法
 - 🔗 **FathestFirst** — K中心点算法
 - 🔗 **OPTICS** — 基于密度的另一个算法
 - 🔗 **Cobweb** — 概念聚类算法
 - 🔗 **sIB** — 基于信息论的聚类算法，不支持分类属性
 - 🔗 **XMeans** — 能自动确定簇个数的扩展K均值算法，不支持分类属性

参数设置

- ❖ 聚类模式
 - 🔗 使用训练集 (Use training set) — 报告训练对象的聚类结果和分组结果
 - 🔗 使用附加的检验集 (Supplied test set) — 报告训练对象的聚类结果和附加的检验对象的分组结果
 - 🔗 百分比划分 (Percentage split) — 报告全部对象的聚类结果、训练对象的聚类结果，以及检验对象的分组结果
 - 🔗 监督评估 (Classes to clusters evaluation) — 报告训练对象的聚类结果和分组结果、类/簇混淆矩阵和错误分组信息
- ❖ **SimpleKMeans**重要参数
 - 🔗 **N** — 簇个数
- ❖ **DBScan**重要参数
 - 🔗 **E** — Eps
 - 🔗 **M** — MinPts

结果分析

❖ 文字分析

SimpleKMeans

- ❖ 非监督模式：运行信息、KMeans结果（迭代次数、SSE、簇中心）、检验对象的分组信息
- ❖ 监督模式：运行信息、KMeans结果（迭代次数、SSE、簇中心）、类/簇混淆矩阵、错误分组的对象个数和比例
- ❖ 簇中心：对于数值属性为均值，对于分类属性为众数

DBScan

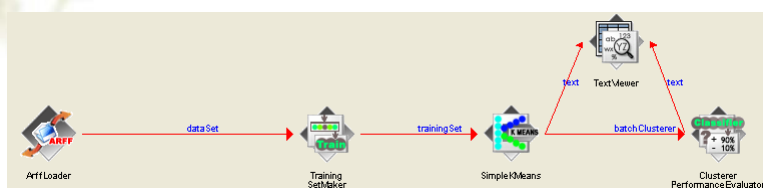
- ❖ 非监督模式：运行信息、DBScan结果（迭代次数、各个训练对象的分组信息）、检验对象的分组信息
- ❖ 监督模式：运行信息、DBScan结果（迭代次数、各个训练对象的分组信息）、类/簇混淆矩阵、错误分组的对象个数和比例

❖ 图形分析（必须将store clusters for visualization勾上）

- ❖ 可视化簇指派 (Visualize cluster assignments): 2D散布图，能够可视化类/簇混淆矩阵

知识流界面运行

❖ 没有评估信息



❖ 产生评估信息



WEKA小结

- ❖ 数据预处理
 - ☞ Explorer – Preprocess:
 - ☞ Explorer – Select attributes: 还可以在Preprocess页面使用属性选择方法
- ❖ 数据可视化
 - ☞ Explorer – Visualize: 二维散布图
- ❖ 分类预测
 - ☞ Explorer – Classify:
 - ☞ Experimenter: 比较多个算法的性能
 - ☞ KnowledgeFlow: 批量/增量学习模式
- ❖ 关联分析
 - ☞ Explorer – Associate:
- ❖ 聚类分析
 - ☞ Explorer – Cluster:

9、扩展Weka

- ❖ 为什么要扩展Weka?
 - ☞ 需要加入第三方的构件
 - ☞ 需要加入自己设计或改进的算法
 - ☞ 需要将Weka整合到实际的应用系统中
- ❖ 要点
 1. 重新编译Weka
 2. 加入新算法（第三方、自己设计或改进）
 3. 在自己的Java程序中使用Weka

9.1、重新编译Weka

1. 下载并安装JDK和JRE环境
(<http://java.sun.com/javase/downloads/index.jsp>)
2. 下载并安装WEKA软件
(http://www.cs.waikato.ac.nz/ml/weka/index_downloading.html)
3. 下载并安装开发环境Eclipse
4. 重新编译weka
 - ① 解压WEKA安装目录中的weka-src.jar到一个新建目录 weka-src下。
 - ② 打开Eclipse, “File”菜单 - “New”项目- 选择“Java Project”。“Project name”写weka。点击下一步。
 - ③ 设置libraries – Add External Jars。
 - ④ 从weka-src里面复制源代码。
 - ⑤ 运行weka.gui.GUIChooser。

9.2、加入新算法

1. 从weka中文站下载FuzzyCMeans.java
2. 复制到weka.clusterers包中
3. 修改FuzzyCMeans.java, 改正错误代码
4. 修改weka.gui.GenericObjectEditor.props, 在#Lists the Clusterers I want to choose from的weka.clusterers.Clusterer=下加入:
weka.clusterers.FuzzyCMeans
5. 重新编译, 运行, 可以在weka的Explorer界面上的Cluster选项卡中找到刚刚新添加的FuzzyCMeans算法
6. 修改FuzzyCMeans.java中的函数getCapabilities(), 以激活FuzzyCMeans算法
7. 重新编译, 运行

9.3、在自己的程序中使用Weka

- ❖ 开发过程中常用的weka组件：
Instances---你的数据
Filter---用于预处理数据
Classifier/Clusterer---从预处理后的数据上建立
Evaluating---评价Classifier/Clusterer的优劣
Attribute Selection---从你的数据中去掉不相关的属性
- ❖ 下面介绍如何在Java程序中使用以上组件。

Instances

ARFF File
Pre 3.5.5 and 3.4.x

- ❖ 直接读入一个ARFF文件并设置类别属性

```
import weka.core.Instances;
import java.io.BufferedReader;
import java.io.FileReader;
...
BufferedReader reader = new BufferedReader( new
    FileReader("/some/where/data.arff"));
Instances data = new Instances(reader);
reader.close();
// setting class attribute
data.setClassIndex(data.numAttributes() - 1);
```


Instances

ARFF File
3.5.5 and newer

- ❖ 使用DataSource类可读入ARFF、CSV以及其它可通过Converter导入的文件

```
import weka.core.converters.ConverterUtils.DataSource;  
...  
DataSource source = new  
    DataSource("/some/where/data.arff");  
Instances data = source.getDataSet();  
// setting class attribute if the data format does not provide  
    this information  
if (data.classIndex() == -1)  
    data.setClassIndex(data.numAttributes() - 1);
```

Option handling

- ❖ Weka通过以下两个方法来设置和获取参数选项

```
void setOptions(String[] options)  
String[] getOptions()
```

- ❖ 有多种方式设置选项

- ✓ 手工生成一个字符串

```
String[] options = new String[2];  
options[0] = "-R";  
options[1] = "1";
```

- ✓ 用splitOptions方法将一个命令行串变成字符串数组

```
String[] options = weka.core.Utils.splitOptions("-R 1");
```

Filter

- ❖ 一个过滤器有两个重要性质
 - **supervised** 或 **unsupervised**
 - **attribute-based** 或 **instance-based**
- ❖ 大多数过滤器都实现了 **OptionHandler** 接口，这意味着你可以用 **String** 数组来设置选项，而无需用 **set-**方法。

例如，假设你要删除数据集的第一个实例，你可以执行

```
weka.filters.unsupervised.attribute.Remove -R 1
```

- ❖ 设 **data** 为一个 **Instances** 对象，则可按以下方式创建和应用一个 **Filter**

```
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;
...
String[] options = new String[2];
options[0] = "-R"; // "range"
options[1] = "1"; // first attribute
Remove remove = new Remove(); // new instance of filter
remove.setOptions(options); // set options
remove.setInputFormat(data); // inform filter about dataset
**AFTER** setting options
Instances newData = Filter.useFilter(data, remove);
```

Building a Classifier

Batch

- ❖ 使用 `buildClassifier(Instances)` 方法在给定的数据集 `Instances` 上建立分类器

```
import weka.classifiers.trees.J48;  
  
...  
String[] options = new String[1];  
options[0] = "-U"; // unpruned tree  
J48 tree = new J48(); // new instance of tree  
tree.setOptions(options); // set the options  
tree.buildClassifier(data); // build classifier
```

Building a Classifier

Incremental

- ❖ 实现了 `weka.classifiers.UpdateableClassifier` 接口的分类器可以被增量地训练，这样节省了内存，因为数据无需一次性地全部装入内存。
- ❖ 训练增量分类器的过程：
 - 🔗 Call `buildClassifier(Instances)` with the structure of the dataset (may or may not contain any actual data rows).
 - 🔗 Subsequently call the `updateClassifier(Instance)` method to feed the classifier one by one.

❖ 示例

```
// load data
ArffLoader loader = new ArffLoader();
loader.setFile(new File("/some/where/data.arff"));
Instances structure = loader.getStructure();
structure.setClassIndex(structure.numAttributes() - 1);
// train
NaiveBayes NaiveBayesUpdateable nb =
    new NaiveBayesUpdateable();
nb.buildClassifier(structure);
Instance current;
while ((current = loader.getNextInstance(structure)) != null)
    nb.updateClassifier(current);
```

Evaluating Cross-validation

❖ 示例

```
import weka.classifiers.Evaluation;
import java.util.Random;

...
Evaluation eval = new Evaluation(newData);
eval.crossValidateModel(tree, newData, 10, new
Random(1));
```

Evaluating

Train/test set

❖ 示例

```
import weka.core.Instances;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;

...
Instances train = ... // from somewhere
Instances test = ... // from somewhere
// train classifier
Classifier cls = new J48();
cls.buildClassifier(train);
// evaluate classifier and print some statistics
Evaluation eval = new Evaluation(train);
eval.evaluateModel(cls, test);
System.out.println(eval.toSummaryString("\nResults\n==
====\n", false));
```

Statistics

● 从evaluation中获取结果的一些方法:

✓ **nominal class**

correct() - number of correctly classified instances (see also incorrect())

pctCorrect() - percentage of correctly classified instances (see also pctIncorrect())

kappa() - Kappa statistics

✓ **numeric class**

correlation Coefficient() - correlation coefficient

✓ **general**

meanAbsoluteError() - the mean absolute error

rootMeanSquaredError() - the root mean squared error

unclassified() - number of unclassified instances

pctUnclassified() - percentage of unclassified instances

Classifying instances

- ❖ 以下示例装入未标识类别的数据
/some/where/unlabeled.arff，并用已训练好的分类器
tree对每条数据进行标识，最后结果保存为
/some/where/labeled.arff

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import weka.core.Instances;

...
// load unlabeled data
Instances unlabeled = new Instances( new
BufferedReader( new
FileReader("/some/where/unlabeled.arff")));
// set class attribute
unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
```

```
// create copy Instances
labeled = new Instances(unlabeled);
// label instances
for (int i = 0; i < unlabeled.numInstances(); i++) {
    double clsLabel =
        tree.classifyInstance(unlabeled.instance(i));
    labeled.instance(i).setClassValue(clsLabel);
}
// save labeled data
BufferedWriter writer = new BufferedWriter( new
    FileWriter("/some/where/labeled.arff"));
writer.write(labeled.toString());
writer.newLine();
writer.close();
```

❖ Note on nominal classes:

If you're interested in the distribution over all the classes, use the method `distributionForInstance(Instance)`. This method returns a double array with the probability for each class.

The returned **double value** from `classifyInstance` (or the index in the array returned by `distributionForInstance`) is just the index for the string values in the attribute. I.e., if you want the string representation for the above returned class label `clsLabel`, then you can, e.g., print it like this:

```
System.out.println(clsLabel + " -> " +  
unlabeled.classAttribute().value((int) clsLabel));
```

Building a Clusterer

Batch

- ❖ 和分类器用 `buildClassifier(Instances)` 建立类似，聚类(器)用 `buildClusterer(Instances)` 建立。下列代码片段(**code snippet**)实现了EM聚类算法，最大迭代次数为100

```
import weka.clusterers.EM;  
  
...  
String[] options = new String[2];  
options[0] = "-I"; // max. iterations  
options[1] = "100";  
EM clusterer = new EM(); // new instance of clusterer  
clusterer.setOptions(options); // set the options  
clusterer.buildClusterer(data); // build the clusterer
```


Building a Clusterer

Incremental

- ❖ 实现了 `weka.clusterers.UpdateableClusterer` 接口的聚类器可以进行增量聚类。
- ❖ 训练增量式聚类器的过程：
 - ✓ Call `buildClusterer(Instances)` with the structure of the dataset (may or may not contain any actual data rows).
 - ✓ Subsequently call the `updateClusterer(Instance)` method to feed the clusterer one by one.
 - ✓ Call `updateFinished()` after all `Instance` objects have been processed, for the clusterer to perform additional computations.

❖ 示例

```
// load data
ArffLoader loader = new ArffLoader();
loader.setFile(new File("/some/where/data.arff"));
Instances structure = loader.getStructure();
// train
Cobweb Cobweb cw = new Cobweb();
cw.buildClusterer(structure);
Instance current;
while ((current = loader.getNextInstance(structure)) != null)
    cw.updateClusterer(current);
cw.updateFinished();
```

Evaluating a clusterer

- ❖ 可用 **ClusterEvaluation** 类评价一个聚类器。以下代码片段输出了算法发现的类的个数。

```
import weka.clusterers.ClusterEvaluation;
import weka.clusterers.Clusterer;
...
ClusterEvaluation eval = new ClusterEvaluation();
Clusterer clusterer = new EM(); //new clusterer instance
clusterer.buildClusterer(data); // build clusterer
eval.setClusterer(clusterer); // the cluster to evaluate
eval.evaluateClusterer(newData); // data to evaluate the
clusterer on
System.out.println("# of clusters: " + eval.getNumClusters());
```

Clustering instances

- ❖ **public int clusterInstance**(Instance instance)
throws java.lang.Exception
 - 🔗 Classifies a given instance. Either this or `distributionForInstance()` needs to be implemented by subclasses.
 - 🔗 **Parameters:**
 - ❖ instance - the instance to be assigned to a cluster
 - 🔗 **Returns:**
 - ❖ the number of the assigned cluster as an integer
- ❖ The only difference to classification is the method name. Instead of `classifyInstance(Instance)` it is now `clusterInstance(Instance)`. The method for obtaining the distribution is still the same, i.e., `distributionForInstance(Instance)`.

Attribute selection

- ❖ 可以采用元分类器(meta-classifier)和过滤器(filter)来实现属性选择。

- ❖ **Meta-Classifer**

The following meta-classifier performs a pre-processing step of attribute selection before the data gets presented to the base classifier (in the example here, this is J48).

```
Instances data = ... // from somewhere
AttributeSelectedClassifier classifier = new
AttributeSelectedClassifier();
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);
J48 base = new J48();
classifier.setClassifier(base);
classifier.setEvaluator(eval);
classifier.setSearch(search); // 10-fold cross-validation
Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier, data, 10, new
Random(1));
System.out.println(evaluation.toSummaryString());
```

❖ Filter

- ❖ The filter approach is straightforward: after setting up the filter, one just filters the data through the filter and obtains the reduced dataset.

```
Instances data = ... // from somewhere
AttributeSelection filter = new AttributeSelection();
// package weka.filters.supervised.attribute!
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);
filter.setEvaluator(eval);
filter.setSearch(search);
filter.setInputFormat(data); // generate new data
Instances newData = Filter.useFilter(data, filter);
```

Java中导入weka类库

- ❖ 在命令行方式中调用weka类库，只需要引入weka的jar包，即把jar加到classpath下。
- ❖ Eclipse中开发配置:右键单击项目名，Build Path → Add External Archives，选weka.jar。