WEKA
The University
of Waikato

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

WEKA in the Ecosystem for Scientific Computing

Contents

- Part 1: Introduction to WEKA
- Part 2: WEKA & Octave
- Part 3: WEKA & R
- Part 4: WEKA & Python
- Part 5: WEKA & Hadoop

For this presentation, we used Ubuntu 13.10 with weka-3-7-11.zip extracted in the user's home folder. All commands were executed from the home folder.

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Part 1:
Introduction to WEKA

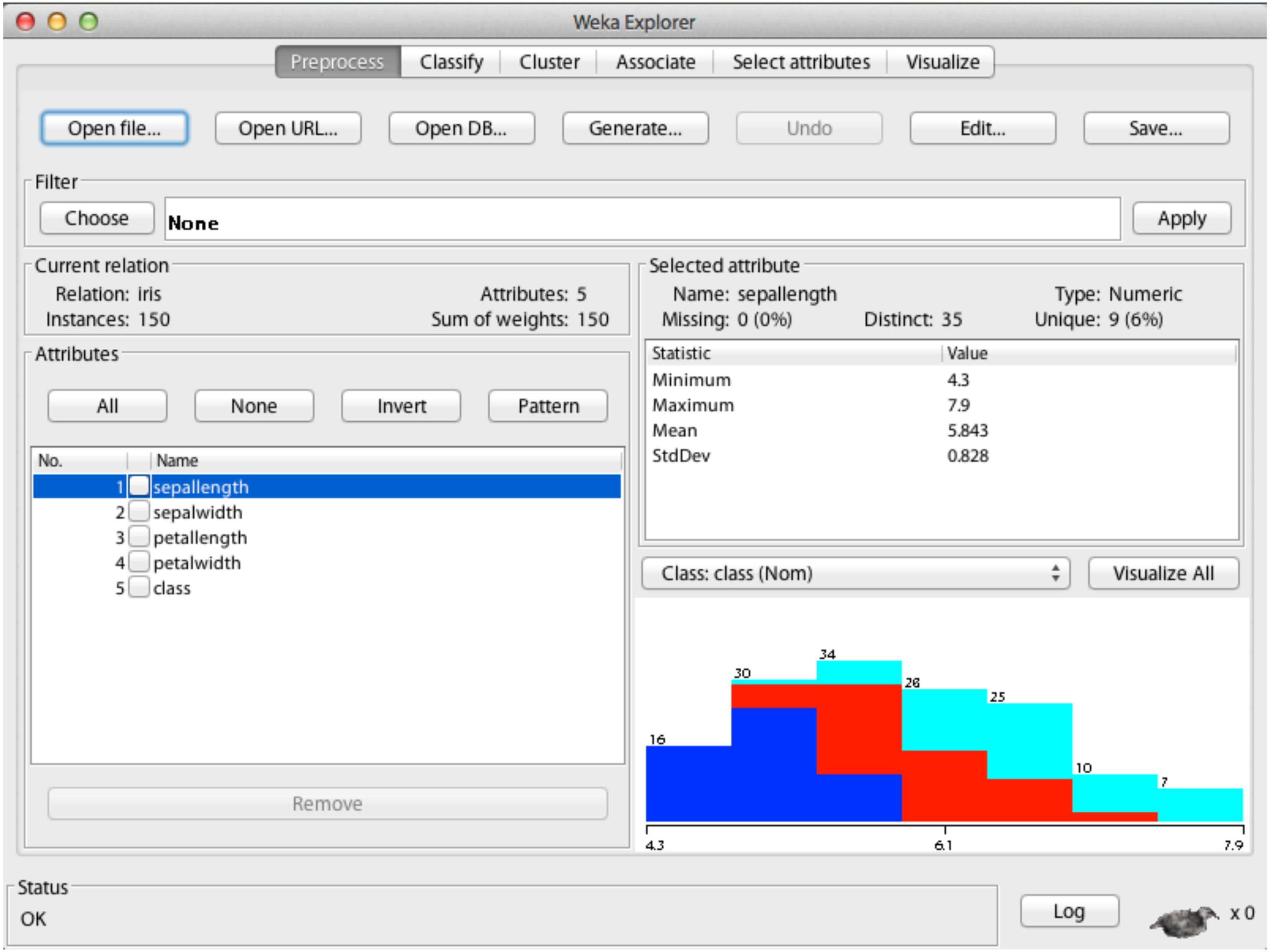
What's WEKA?

- WEKA is a library containing a large collection of machine learning algorithms, implemented in Java
- Main types of learning problems that it can tackle:
  - Classification: given a labelled set of observations, learn to predict labels for new observations
  - Regression: numeric value instead of label
  - Attribute selection: find attributes of observations that are important for prediction
  - Clustering: no labels, just identify groups of similar observations (clusters)
- There is also some support for association rule mining and (conditional) density estimation

How to use it via built-in options?

- WEKA has three built-in graphical user interfaces:
  - Explorer: still the most popular interface for batch data processing; tab-based interface to algorithms
  - Knowledge Flow: users lay out and connect widgets representing WEKA components
  - Experimenter: enables large scale comparison of predictive performance of learning algorithms
- WEKA also has a command-line interface and its functionality can be accessed through the OS shell
- Only Knowledge Flow and command-line interface enable incremental processing of data

Explorer



Filter: None

Relation: iris
Attributes: 5
Instances: 150
Sum of weights: 150

No.		Name
1	□	sepallength
2	□	sepalwidth
3	□	petallength
4	□	petalwidth
5	□	class

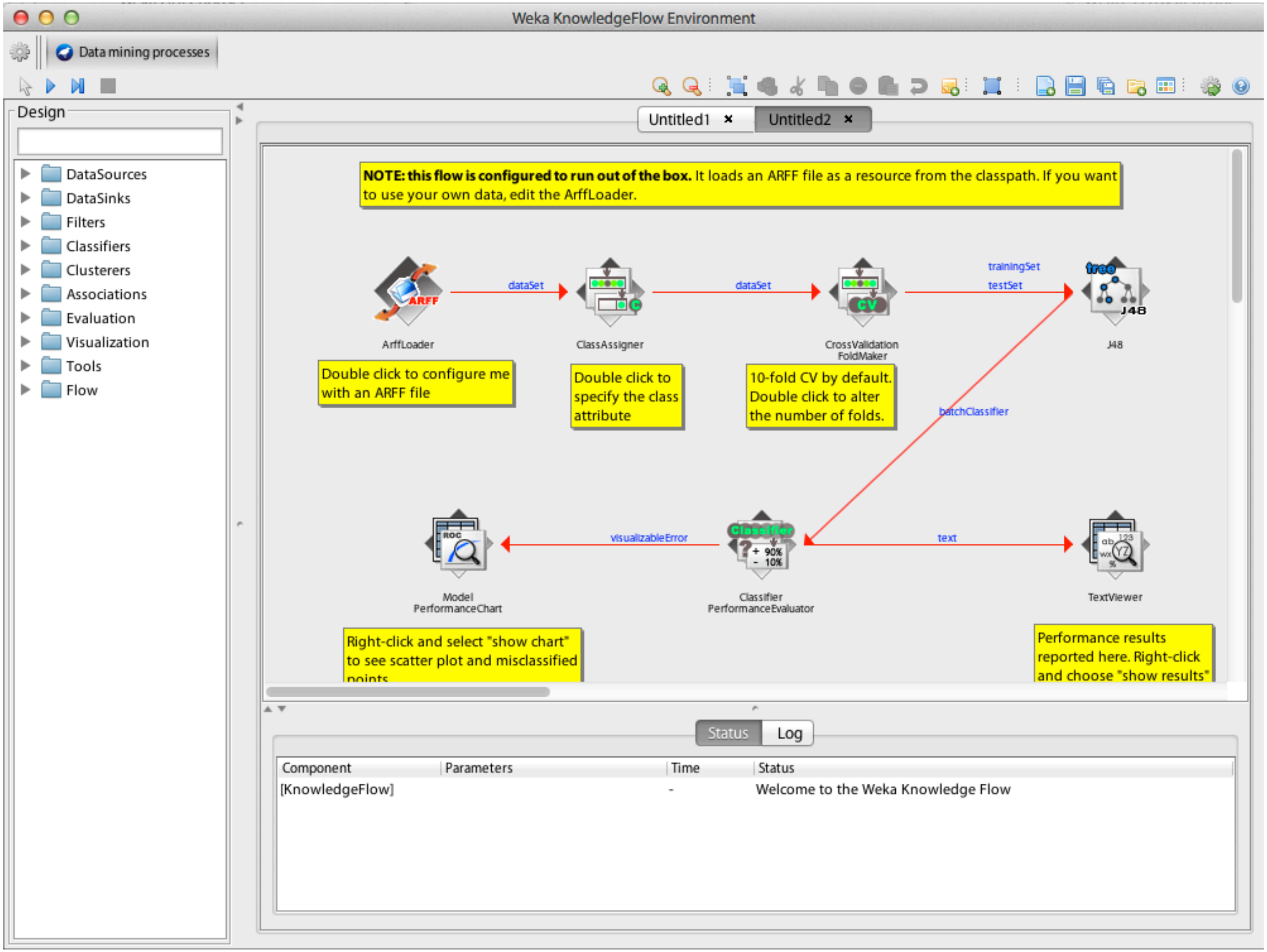
Name: sepallength
Type: Numeric
Missing: 0 (0%)
Distinct: 35
Unique: 9 (6%)

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

Class: class (Nom)

Status: OK

Knowledge Flow



NOTE: this flow is configured to run out of the box. It loads an ARFF file as a resource from the classpath. If you want to use your own data, edit the ArffLoader.

Double click to configure me with an ARFF file

Double click to specify the class attribute

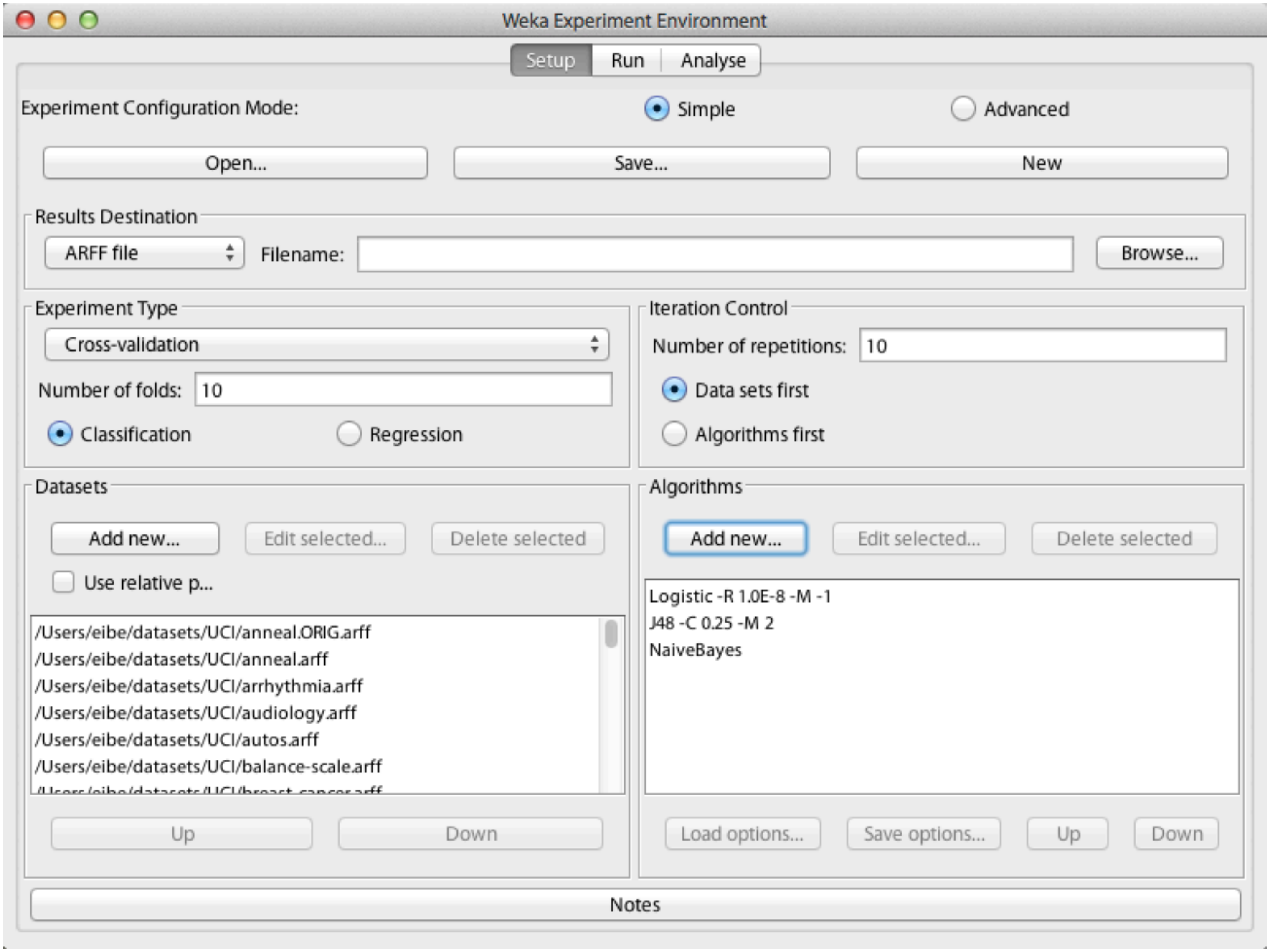
10-fold CV by default. Double click to alter the number of folds.

Right-click and select "show chart" to see scatter plot and misclassified points

Performance results reported here. Right-click and choose "show results"

Component	Parameters	Time	Status
[KnowledgeFlow]		-	Welcome to the Weka Knowledge Flow

Experimenter



Experiment Configuration Mode: ● Simple ○ Advanced

Results Destination

ARFF file Filename:

Experiment Type

Cross-validation

Number of folds: 10

● Classification ○ Regression

Iteration Control

Number of repetitions: 10

● Data sets first

○ Algorithms first

Datasets

□ Use relative p...

/Users/eibe/datasets/UCI/anneal.ORIG.arff
/Users/eibe/datasets/UCI/anneal.arff
/Users/eibe/datasets/UCI/arrhythmia.arff
/Users/eibe/datasets/UCI/audiology.arff
/Users/eibe/datasets/UCI/autos.arff
/Users/eibe/datasets/UCI/balance-scale.arff
/Users/eibe/datasets/UCI/breast-cancer.arff

Algorithms

Logistic -R 1.0E-8 -M -1
J48 -C 0.25 -M 2
NaiveBayes

How to use it via external interfaces?

- WEKA provides a unified interface to a large collection of learning algorithms and is implemented in Java
- There is a variety of software through which one can make use of this interface
    - Octave/Matlab
    - R statistical computing environment: RWeka
    - Python: python-weka-wrapper
- Other software through which one can access WEKA: Mathematica, SAS, KNIME, RapidMiner

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Part 2:
WEKA & Octave

Octave

- GNU Octave is an open-source version of Matlab (https://www.gnu.org/software/octave/)
- Provides access to Java code through its Java package (http://wiki.octave.org/Java_package)
- Installation on Ubuntu Linux:

```
sudo apt-get install octave
sudo apt-get install octave-java
```

- In Octave, add WEKA to the Java CLASSPATH:

```
javaaddpath("weka-3-7-11/weka.jar")
```

- Check CLASSPATH:

```
javaclasspath()
```

Loading a dataset and building a tree

- Load and output a dataset in WEKA's ARFF format by creating an ArffLoader object:

```
l = javaObject("weka.core.converters.ArffLoader")
l.setFile(javaObject("java.io.File",
           "weka-3-7-11/data/iris.arff"))
d = l.getDataSet
d.toString
```

- Build and output a J48 decision tree:

```
c = javaObject("weka.classifiers.trees.J48")
d.setClassIndex(d.numAttributes - 1)
c.buildClassifier(d)
c.toString
```

Evaluation and data saving

- Evaluate how well tree will predict, using 10-fold stratified cross-validation to estimate classification error, etc.:

```
e = javaObject("weka.classifiers.Evaluation", d)
e.crossValidateModel(c, d, 10,
                     javaObject("java.util.Random", 1),
                     javaArray("java.lang.Object",0))
e.toSummaryString
```

- Save data in Matlab format, load it back, and plot it:

```
s = javaObject("weka.core.converters.MatlabSaver")
s.setFile(javaObject("java.io.File", "iris.data"))
s.setInstances(d)
s.writeBatch
m = load("iris.data")
scatter(m(:, 3), m(:, 4), 20, m(:, 5))
```

Filtering and data conversion

- Build classifier from reduced dataset:

```
f = javaObject("weka.filters.unsupervised.
                     attribute.Remove")
f.setAttributeIndices("1-2")
f.setInputFormat(d)
rD = javaMethod("useFilter",
                "weka.filters.Filter", d, f)
c.buildClassifier(rD)
c.toString
```

- Turn reduced data into Matlab matrix:

```
rM = zeros(rD.numInstances, rD.numAttributes)
for i = 1:rD.numInstances
  for j = 1:rD.numAttributes
    rM(i,j) = rD.instance(i - 1).value(j - 1)
  end
end
```

Storing and visualizing predictions

- Store predictions for reduced dataset in a matrix:

```
p = zeros(rD.numInstances, rD.numClasses)
for i = 1:rD.numInstances
  dist = c.distributionForInstance(rD.instance(i - 1))
  for j = 1:rD.numClasses
    p(i,j) = dist(j)
  end
end
```

- Plot data using colors based on predicted probabilities:

```
scatter(rM(:,1), rM(:,2), 20, p)
```

Generating predictions for a grid of points

```
[x, y] = meshgrid(1:.1:7, 0:.1:2.5)
x = x(:)
y = y(:)
gM = [x y]
save grid.data gM -ascii
l = javaObject("weka.core.converters.MatlabLoader")
l.setFile(javaObject("java.io.File","grid.data"))
gD = l.getDataSet
gD.insertAttributeAt(rD.attribute(2), 2)
gD.setClassIndex(2)
p = zeros(gD.numInstances, gD.numClasses)
for i = 1:gD.numInstances
  dist = c.distributionForInstance(gD.instance(i - 1))
  for j = 1:gD.numClasses
    p(i, j) = dist(j)
  end
end
scatter(gM(:,1), gM(:,2), 20, p)
```

Clustering and visualizing data

```
f = javaObject("weka.filters.unsupervised.
                      attribute.Remove")
f.setAttributeIndices("last")
f.setInputFormat(d)
rD = javaMethod("useFilter","weka.filters.Filter", d, f)
c = javaObject("weka.clusterers.SimpleKMeans")
c.setNumClusters(3)
c.buildClusterer(rD)
c.toString
a = zeros(rD.numInstances, 1)
for i = 1:rD.numInstances
  a(i) = c.clusterInstance(rD.instance(i - 1))
end
scatter(m(:,3), m(:,4), 20, a)
```

Finding the most important predictors

```
as = javaObject("weka.attributeSelection.
                   AttributeSelection")
s = javaObject("weka.attributeSelection.GreedyStepwise")
s.setSearchBackwards(true)
as.setSearch(s)
e = javaObject("weka.attributeSelection.
                          WrapperSubsetEval")
e.setClassifier(javaObject("weka.
                            classifiers.trees.J48"))
as.setEvaluator(e)
as.SelectAttributes(d)
as.toResultsString
```

Build a classifier with attribute selection

- Build a tree based on selected attributes:

```
c = javaObject("weka.classifiers.meta.
                      AttributeSelectedClassifier")
c.setEvaluator(e)
c.setSearch(s)
c.setClassifier(javaObject("weka.
                            classifiers.trees.J48"))
c.buildClassifier(d)
c.toString
```

- Estimate performance of model with attribute selection:

```
e = javaObject("weka.classifiers.Evaluation", d)
e.crossValidateModel(c, d, 10,
                   javaObject("java.util.Random", 1),
                   javaArray("java.lang.Object",0))
e.toSummaryString
```

Using code from a WEKA package

- WEKA 3.7 has a package management system through which a lot of additional packages are available
- These packages are in separate Java .jar archives, not in the main weka.jar archive
- Need to load all these packages into the Octave CLASSPATH, so that they can be used from Octave:

```
javaMethod("loadPackages",
  "weka.core.WekaPackageManager", false, true, false)
```

- Should check CLASSPATH afterwards:

```
javaclasspath()
```

WEKA
The University
of Waikato

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Part 3:
WEKA & R

R

- Open-source R system for statistical computing implements the S language developed at Bell labs (http://www.r-project.org/)
- Provides access to Weka through RWeka package (http://cran.r-project.org/package=RWeka)
- Installation on Ubuntu Linux (tried with Ubuntu 13.10):

```
sudo apt-get install r-base
```

- In R, install RWeka package (includes latest WEKA):

```
install.packages("RWeka", dependencies = TRUE)
```

- Once installed, start R, and load the package into R:

```
library(RWeka)
```

Using WEKA from R

- Read ARFF file into R data frame and plot it:

```
d <- read.arff(file("weka-3-7-11/data/iris.arff"))
plot(d, col=c("red","blue","green")[d$class])
```

- Build and output decision tree using built-in J48 wrapper:

```
c <- J48(class ~., d)
c
```

- Install and use partykit package for tree visualization:

```
install.packages("partykit", dependencies = TRUE)
library(partykit)
plot(c)
```

- Run 10-fold cross-validation using tree learner:

```
evaluate_Weka_classifier(c, numFolds = 10)
```

Accessing arbitrary classifiers and filters

- Can access any classifier in WEKA:

```
NB <- make_Weka_classifier("weka.classifiers.bayes.
                             NaiveBayes")
NB
NB(class ~., d)
```

- List scheme options and change them using control:

```
WOW(NB)
NB(class ~., d, control = Weka_control(D = TRUE))
```

- A similar process works for filters:

```
Remove <- make_Weka_filter("weka.filters.
                 unsupervised.attribute.Remove")
WOW(Remove)
rD <- Remove(~., d, control = Weka_control(R = "1,2"))
rD
```

Storing and visualizing predictions

- Obtain predicted class probabilities and plot them:

```
c <- J48(class ~., rD)
p <- predict(c, rD, c("probability"))
plot(rD[1:2], col = rgb(p))
```

- Generate grid and plot predicted probabilities:

```
gD <- expand.grid(petallength = seq(1, 7, 0.1),
 petalwidth = seq(0, 2.5, 0.1), class =
 c("Iris-setosa","Iris-versicolor", "Iris-virginica"))
p <- predict(c, gD, c("probability"))
plot(gD[1:2], col = rgb(p))
```

Clustering data

- Build clustering model:

```
rD <- Remove(~., d, control = Weka_control(R = "last"))
c <- SimpleKMeans(rD, control = Weka_control(N = 3))
c
```

- Visualize cluster assignments:

```
p <- predict(c, rD, c("membership"))
plot(rD[3:4], col = rgb(p))
```

Attribute selection

- Rank the predictors:

```
InfoGainAttributeEval(class ~., d)
```

- Attribute subset selection can be applied as part of learning a classifier:

```
AttributeSelectedClassifier = make_Weka_classifier("weka.
           classifiers.meta.AttributeSelectedClassifier")
c = AttributeSelectedClassifier(class ~., d,
       control = Weka_control(W = ".J48",
                   E = ".WrapperSubsetEval -B .J48",
                   S = ".GreedyStepwise -B"))
evaluate_Weka_classifier(c, numFolds = 10)
```

- We can also make a filter for attribute subset selection using weka.filters.supervised.attribute.AttributeSelection

Text classification and performance plots

```
FilteredClassifier = make_Weka_classifier("weka.classifiers.
                        meta.FilteredClassifier")
d <- read.arff("weka-3-7-11/data/ReutersCorn-train.arff")
fc <- FilteredClassifier(`class-att` ~., d,
       control = Weka_control(F = ".StringToWordVector",
                            W = ".NaiveBayesMultinomial"))
td <- read.arff("weka-3-7-11/data/ReutersCorn-test.arff")
p = predict(fc, td, "probability")[,"1"]
labels = td["class-att"]
install.packages("ROCR")
library(ROCR)
pred <- prediction(p, labels)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
perf <- performance(pred, "sens", "spec")
plot(perf)
perf <- performance(pred, "cal")
plot(perf)
```

WPM command for package management

- Refresh cache of WEKA packages:

```
WPM("refresh-cache")
```

- List packages:

```
WPM("list-packages", "installed")
WPM("list-packages", "available")
```

- Print package info:

```
WPM("package-info", "repository", "XMeans")
```

- Install and load package:

```
WPM("install-package", "XMeans")
WPM("load-package", "XMeans")
```

Using R from WEKA

- RPlugin package for WEKA 3.7 provides: (http://weka.sourceforge.net/packageMetaData/RPlugin/index.html)

  ```
  java weka.core.WekaPackageManager -install-package RPlugin
  ```

  - A Knowledge Flow component to execute R scripts
  - A "wrapper" classifier for all MLR algorithms
  - An R console for the Explorer and Knowledge Flow

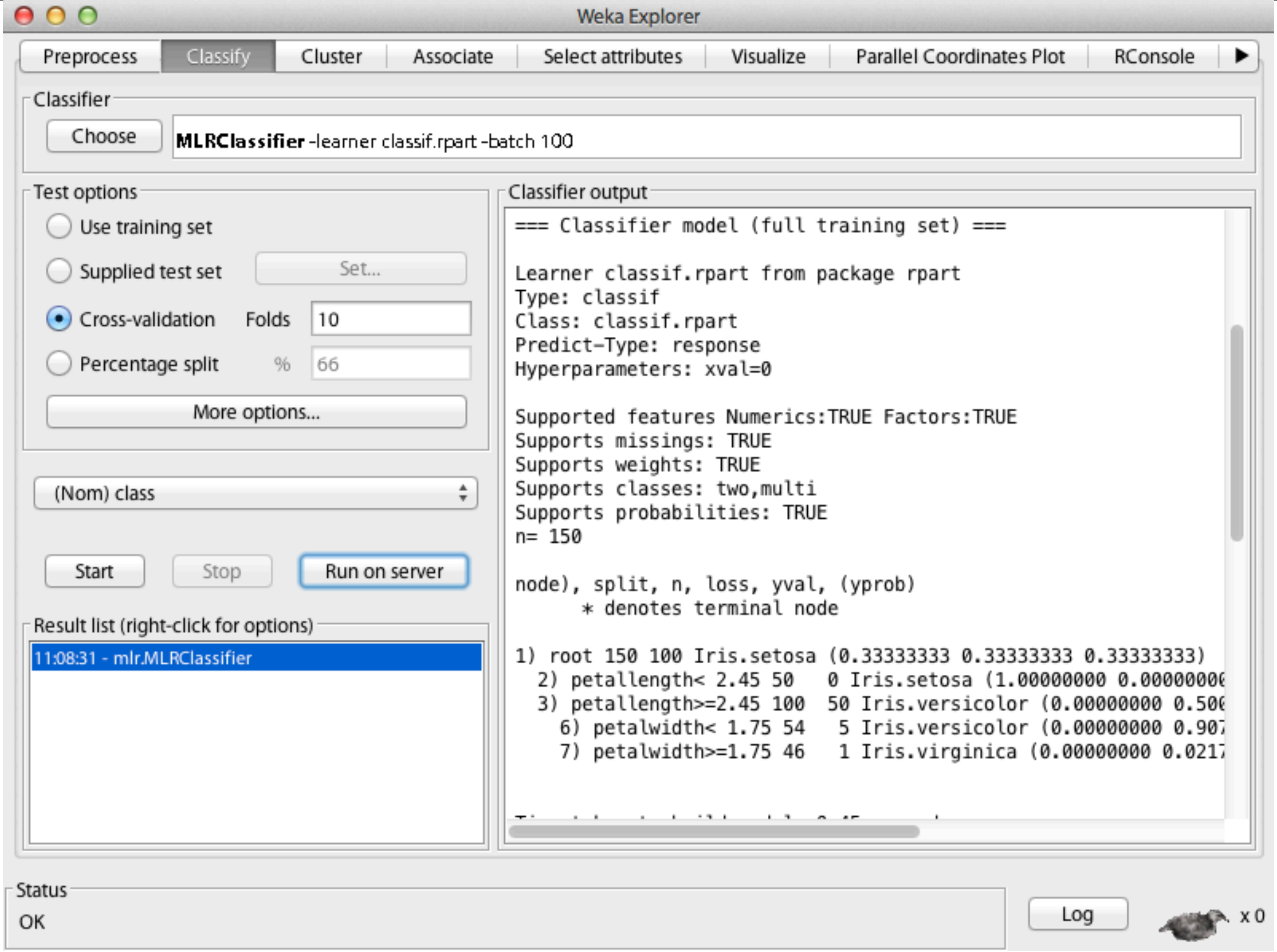
- Requires environment variable R_HOME to be set to the value returned by issuing the following command in R:

  ```
  R.home(component = "home")
  ```

- Set R_LIBS_USER to first value returned by: `.libPaths()`
- rJava package needs to be installed in R:

  ```
  install.packages("rJava")
  ```

Applying an MLR classifier



Classifier: MLRClassifier -learner classif.rpart -batch 100

Test options
- ○ Use training set
- ○ Supplied test set
- ● Cross-validation Folds 10
- ○ Percentage split % 66

(Nom) class

Result list (right-click for options)

11:08:31 - mlr.MLRClassifier

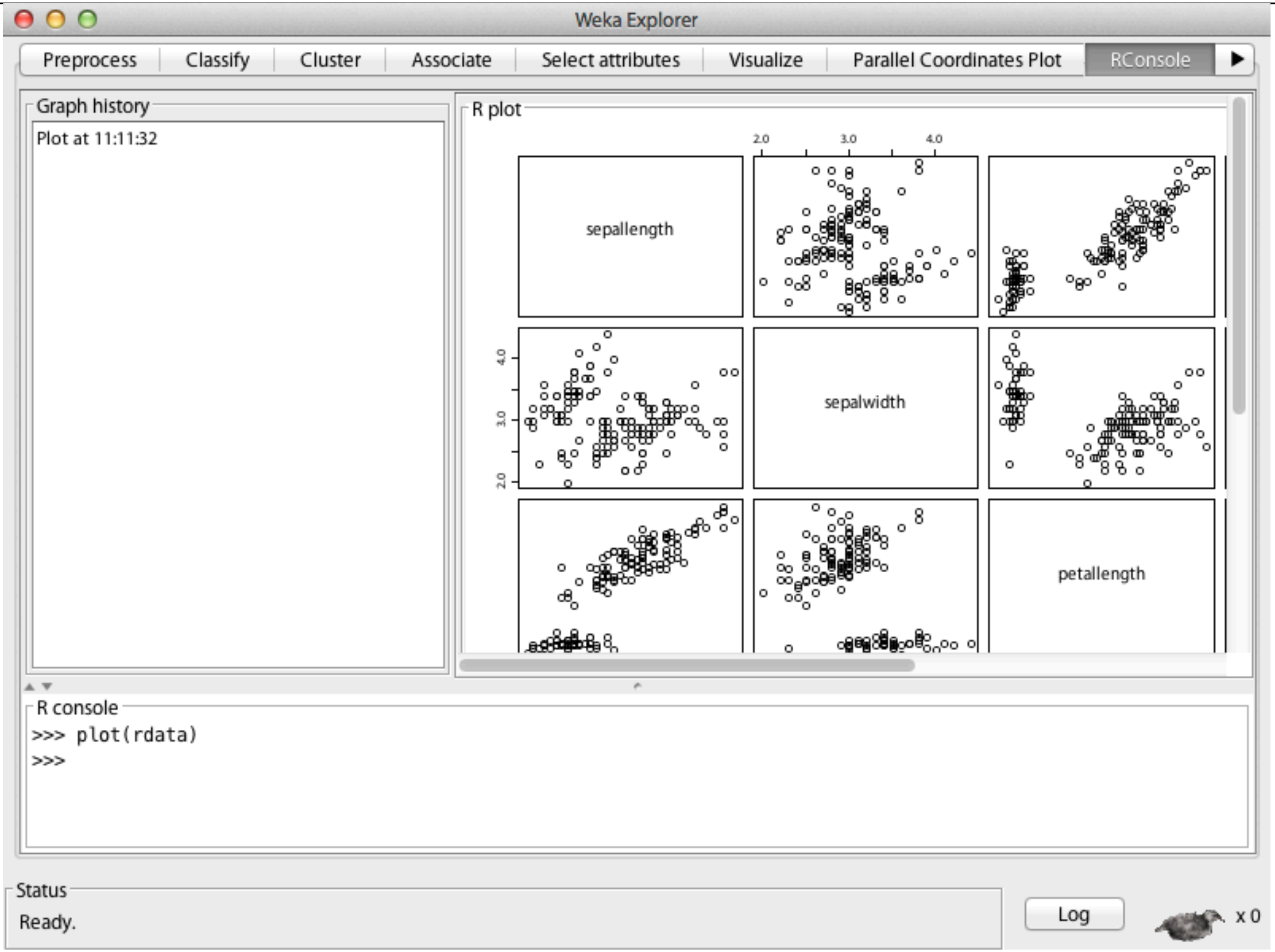
Classifier output

```
=== Classifier model (full training set) ===

Learner classif.rpart from package rpart
Type: classif
Class: classif.rpart
Predict-Type: response
Hyperparameters: xval=0

Supported features Numerics:TRUE Factors:TRUE
Supports missings: TRUE
Supports weights: TRUE
Supports classes: two,multi
Supports probabilities: TRUE
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 Iris.setosa (0.33333333 0.33333333 0.33333333)
  2) petallength< 2.45 50   0 Iris.setosa (1.00000000 0.0000000
  3) petallength>=2.45 100  50 Iris.versicolor (0.00000000 0.500
    6) petalwidth< 1.75 54   5 Iris.versicolor (0.00000000 0.907
    7) petalwidth>=1.75 46   1 Iris.virginica (0.00000000 0.0217
```

Status

OK

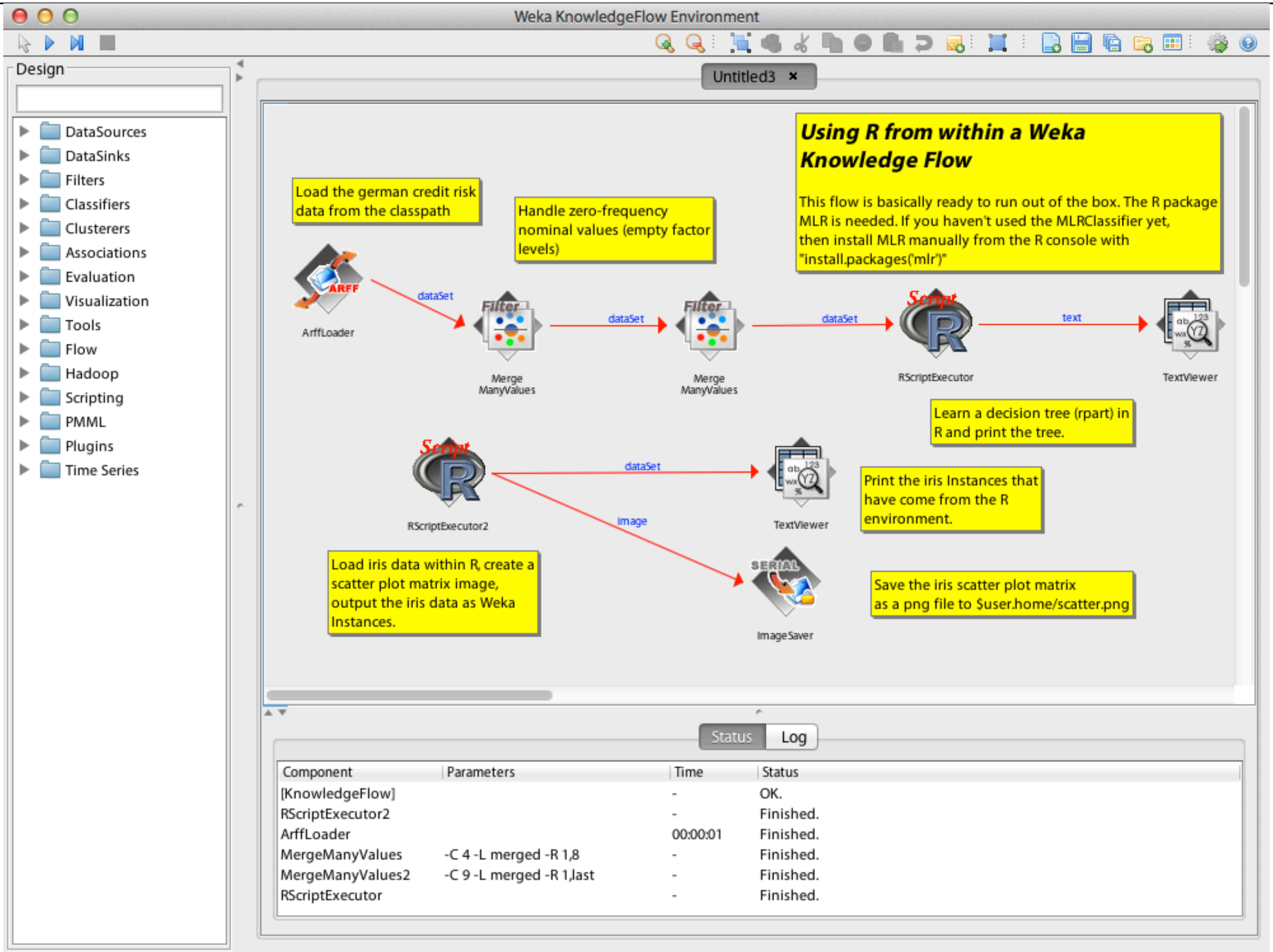
Visualizing data using the R console



R console

```
>>> plot(rdata)
>>>
```

Processing data using an R component



Using R from within a Weka Knowledge Flow

This flow is basically ready to run out of the box. The R package MLR is needed. If you haven't used the MLRClassifier yet, then install MLR manually from the R console with "install.packages('mlr')"

Load the german credit risk data from the classpath

Handle zero-frequency nominal values (empty factor levels)

Learn a decision tree (rpart) in R and print the tree.

Print the iris Instances that have come from the R environment.

Load iris data within R, create a scatter plot matrix image, output the iris data as Weka Instances.

Save the iris scatter plot matrix as a png file to $user.home/scatter.png

Component	Parameters	Time	Status
[KnowledgeFlow]		-	OK.
RScriptExecutor2		-	Finished.
ArffLoader		00:00:01	Finished.
MergeManyValues	-C 4 -L merged -R 1,8	-	Finished.
MergeManyValues2	-C 9 -L merged -R 1,last	-	Finished.
RScriptExecutor		-	Finished.

WEKA

The University of Waikato

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Part 4:
WEKA & Python

Python and WEKA

- WEKA can be used directly from Jython, a Python implementation for the Java Virtual Machine
- However, several important libraries for Python are implemented in native code and not available in Jython
- Fortunately, there is a nice way to run WEKA from Python: http://pythonhosted.org//python-weka-wrapper/
- First, install dev tools, pip and packages for Python:

```
sudo apt-get install python-pip python-numpy
  python-dev python-imaging python-matplotlib
  python-pygraphviz imagemagick
```

- Then, install javabridge and weka wrapper for Python:

```
sudo pip install javabridge python-weka-wrapper
```

Using WEKA from Python

- First, need to start the JVM from Python:

```
import weka.core.jvm as jvm
jvm.start()
```

- We can get help on the commands:

```
help(jvm.start)
```

- Load and print some data in ARFF format:

```
from weka.core.converters import Loader
l = Loader("weka.core.converters.ArffLoader")
d = l.load_file("weka-3-7-11/data/iris.arff")
d.set_class_index(d.num_attributes() - 1)
print(d)
```

Building and evaluating a classifier

- Build and print a decision tree:

```
from weka.classifiers import Classifier
c = Classifier("weka.classifiers.trees.J48")
c.build_classifier(d)
print(c)
```

- Evaluate classifier using cross-validation:

```
from weka.classifiers import Evaluation
from weka.core.classes import Random
e = Evaluation(d)
e.crossvalidate_model(c, d, 10, Random(1))
print(e.percent_correct())
print(e.to_summary())
print(e.to_class_details())
```

Visualize classifier based on filtered data

```
from weka.filters import Filter
r =Filter("weka.filters.unsupervised.attribute.Remove",
 options = ["-R", "1, 2"])
r.set_inputformat(d)
rD = r.filter(d)
c.build_classifier(rD)
import weka.plot.graph as graph
graph.plot_dot_graph(c.graph())

import weka.plot.dataset as pld
pld.scatter_plot(rD, 0, 1, percent=100)
```

Visualize class probabilities

```
r = range(0, rD.num_instances())
x = [rD.get_instance(i).get_value(0) for i in r]
y = [rD.get_instance(i).get_value(1) for i in r]
p = [c.distribution_for_instance(rD.get_instance(i))
          for i in r]
import matplotlib.pyplot as plot
plot.scatter(x, y, 20, p)
plot.show()
```

Plot grid of predictions

```
s0 = rD.get_attribute_stats(0).numeric_stats()
s1 = rD.get_attribute_stats(1).numeric_stats()
r = range(0,101)
x = [s0.min()+(s0.max()-s0.min())*i/100 for i in r]
y = [s1.min()+(s1.max()-s1.min())*i/100 for i in r]
gD = d.template_instances(rD)
from weka.core.dataset import Instance
for i in range(len(x)):
  for j in range(len(y)):
    gD.add_instance(Instance.
            create_instance([x[i], y[j], 0]))
r = range(0, gD.num_instances())
x = [gD.get_instance(i).get_value(0) for i in r]
y = [gD.get_instance(i).get_value(1) for i in r]
p = [c.distribution_for_instance(gD.get_instance(i))
                                  for i in r]
import matplotlib.pyplot as plot
plot.scatter(x, y, 20, p)
plot.show()
```

Cluster data and visualize clusters

```
r =Filter("weka.filters.unsupervised.attribute.Remove",
            options = ["-R", "last"])
r.set_inputformat(rD)
rD = r.filter(rD)
from weka.clusterers import Clusterer
clu = Clusterer("weka.clusterers.SimpleKMeans",
            options = ["-N", "3"])
clu.build_clusterer(rD)
print(clu)

r  =range(0, rD.num_instances())
x = [rD.get_instance(i).get_value(0) for i in r]
y = [rD.get_instance(i).get_value(1) for i in r]
p = [clu.distribution_for_instance(rD.get_instance(i))
          for i in r]
import matplotlib.pyplot as plot
plot.scatter(x, y, 20, p)
plot.show()
```

Attribute selection

```
from weka.attribute_selection import
      ASSearch, ASEvaluation, AttributeSelection
s = ASSearch("weka.attributeSelection.GreedyStepwise",
             options = ["-B"])
e = ASEvaluation("weka.attributeSelection.
                            WrapperSubsetEval",
                 options=["-B", ".J48"])
attS = AttributeSelection()
attS.set_search(s)
attS.set_evaluator(e)
attS.select_attributes(d)
print(attS.to_results_string())
```

Build a classifier with attribute selection

- Build a tree based on selected attributes:

```
c = Classifier("weka.classifiers.meta.
                  AttributeSelectedClassifier",
        options = ["-S", ".GreedyStepwise -B", "-E",
                   ".WrapperSubsetEval -B .J48"])
c.build_classifier(d)
print(c)
```

- Estimate performance of model with attribute selection:

```
from weka.classifiers import Evaluation
from weka.core.classes import Random
e = Evaluation(d)
e.crossvalidate_model(c, d, 10, Random(1))
print(e.to_summary())
```

Managing WEKA packages from Python

```
import weka.core.packages as packages
items = packages.get_all_packages()
for item in items:
  if item.get_name() == "CLOPE":
    print item.get_name(), item.get_url()
packages.install_package("CLOPE")
items = packages.get_installed_packages()
for item in items:
  print item.get_name(), item.get_url()
from weka.clusterers import Clusterer
clu = Clusterer("weka.clusterers.CLOPE")
clu.build_clusterer(rD)
print(clu)
packages.uninstall_package("CLOPE")
items = packages.get_installed_packages()
for item in items:
  print item.get_name(), item.get_url()
```

WEKA
The University
of Waikato

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Part 5:
WEKA & Hadoop

Apache Hadoop

- Java system for distributed storage (HDFS) and computation (MapReduce)
- Useful for storing and processing large datasets that are too large for a single computer
- WEKA 3.7 now has some support for using Hadoop, based on two packages:
  - The distributedWekaBase package has basic infrastructure for distributed computation
  - The distributedWekaHadoop package provides an implementation for Hadoop
- In the following, we will install and use Hadoop on a single computer for simplicity

Setting things up

- Download and install Hadoop package:

```
wget
https://archive.apache.org/dist/hadoop/core/
hadoop-1.2.1/hadoop-1.2.1-bin.tar.gz
tar -xzf hadoop-1.2.1-bin.tar.gz
```

- Need to modify the following configuration files in hadoop-1.2.1/conf/:
  - core-site.xml
  - hdfs-site.xml
  - mapred-site.xml
  - hadoop-env.sh

core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:8020</value>
</property>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/eibe/hadoop/tmp</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/home/eibe/hadoop/data</value>
</property>

</configuration>
```

hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

<property>
  <name>dfs.permissions</name>
  <value>false</value>
</property>

</configuration>
```

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>localhost:8021</value>
</property>

</configuration>
```

Starting Hadoop

- Set the location of JAVA_HOME in hadoop-env.sh:

```
export JAVA_HOME=$(readlink -f /usr/bin/javac |
             sed "s:/bin/javac::")
```

- Install Open SSH and enable password-less login:

```
sudo apt-get install openssh-client openssh-server
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- Format the Hadoop file system:

```
hadoop-1.2.1/bin/hadoop namenode -format
```

- Start Hadoop:

```
hadoop-1.2.1/bin/start-all.sh
```

Setting up WEKA and transferring data

- Set the Java CLASSPATH to point to WEKA:

```
export CLASSPATH=/home/eibe/weka-3-7-11/weka.jar
```

- Install the necessary WEKA packages:

```
java weka.core.WekaPackageManager
        -install-package distributedWekaHadoop
```

- Save some data in CSV format in HDFS:

```
java weka.Run .HDFSSaver -i ~/weka-3-7-11/data/iris.arff
    -dest /users/eibe/input/classification/iris.csv
    -saver "weka.core.converters.CSVSaver -N"
```

- Check that the data is in fact in HDFS:

```
hadoop-1.2.1/bin/hadoop fs
    -cat /users/eibe/input/classification/iris.csv
```

Running some WEKA jobs

- Create an ARFF file with summary information in HDFS:

```
java weka.Run .ArffHeaderHadoopJob
     -input-paths /users/eibe/input/classification
     -output-path /users/eibe/output
     -A sepallength,sepalwidth,petallength,petalwidth,class
     -header-file-name iris.header.arff
```

- Can check on jobs by browsing to: http://localhost:50030
- Check the header file:

```
hadoop-1.2.1/bin/hadoop fs
     -cat /users/eibe/output/arff/iris.header.arff
```

- Compute correlation matrix:

```
java weka.Run .CorrelationMatrixHadoopJob
  -existing-header /users/eibe/output/arff/iris.header.arff
  -class last -input-paths /users/eibe/input/classification
  -output-path /users/eibe/output
```

Building and evaluating classifiers

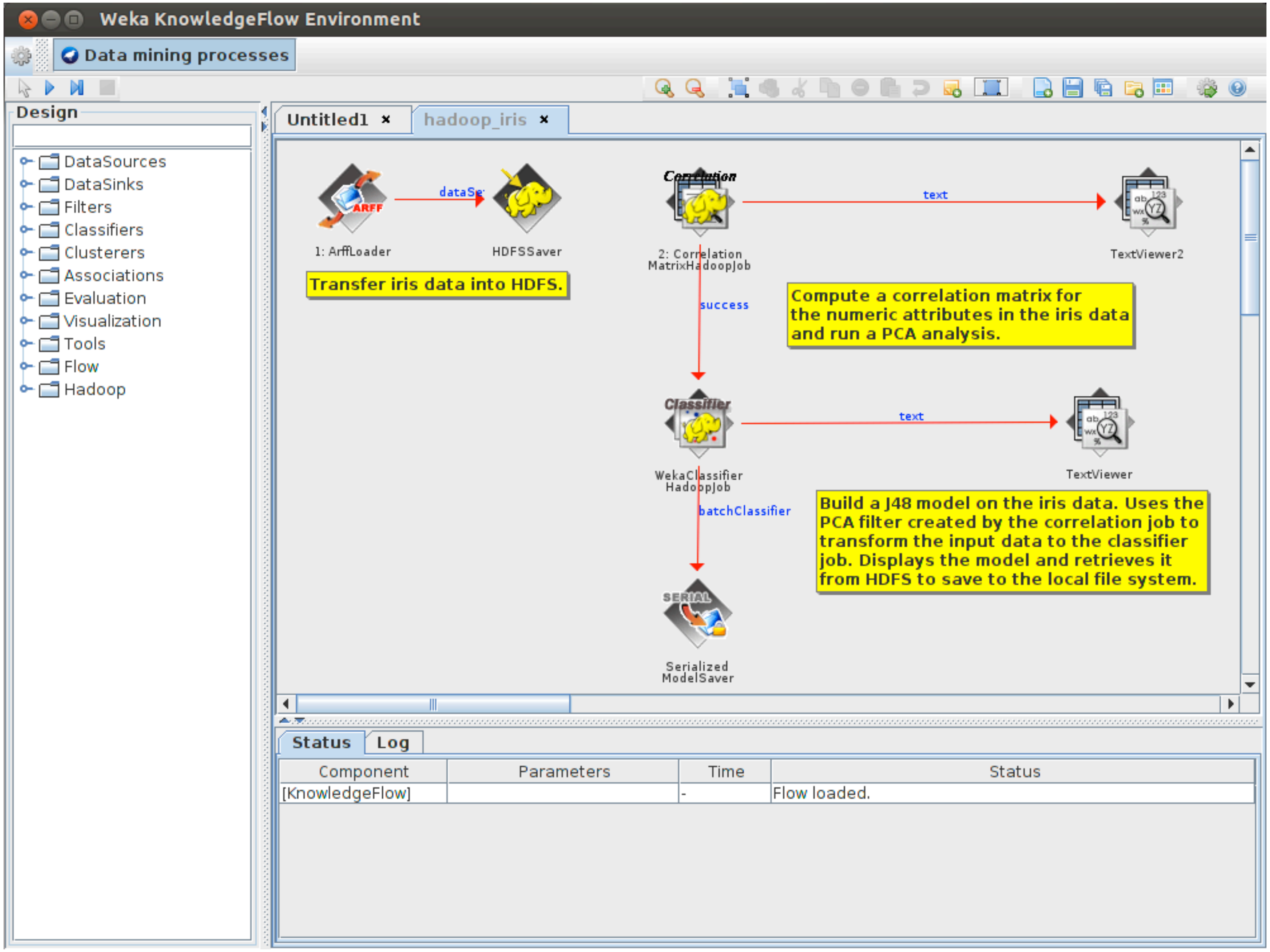
- Build an ensemble of J48 trees (using "dagging"):

```
java weka.Run .WekaClassifierHadoopJob
  -existing-header /users/eibe/output/arff/iris.header.arff
  -class last -input-paths /users/eibe/input/classification
  -output-path /users/eibe/output
  -W weka.classifiers.trees.J48
  -model-file-name J48_dist.model
  -randomized-chunks -num-chunks 10
```

- Evaluate the classifier using cross-validation in Hadoop:

```
java weka.Run .WekaClassifierEvaluationHadoopJob
  -existing-header /users/eibe/output/arff/iris.header.arff
  -class last -input-paths /users/eibe/input/classification
  -output-path /users/eibe/output
  -W weka.classifiers.trees.J48
  -model-file-name J48_dist.model
  -randomized-chunks -num-chunks 10 -num-folds 10
```

Using Hadoop via the Knowledge Flow GUI



Transfer iris data into HDFS.

Compute a correlation matrix for the numeric attributes in the iris data and run a PCA analysis.

Build a J48 model on the iris data. Uses the PCA filter created by the correlation job to transform the input data to the classifier job. Displays the model and retrieves it from HDFS to save to the local file system.

Component	Parameters	Time	Status
[KnowledgeFlow]		-	Flow loaded.

http://www.cs.waikato.ac.nz/ml/weka/xml/examples/hadoop_iris.kfml