THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

WEKA Experimenter Tutorial for Version 3-5-3

David Scuse
Peter Reutemann

June 8, 2006

Contents

1 Introduction

The Weka Experiment Environment enables the user to create, run, modify, and analyse experiments in a more convenient manner than is possible when processing the schemes individually. For example, the user can create an experiment that runs several schemes against a series of datasets and then analyse the results to determine if one of the schemes is (statistically) better than the other schemes.

The Experiment Environment can be run from the command line using the Simple CLI. For example, the following commands could be typed into the CLI to run the `OneR` scheme on the Iris dataset using a basic train and test process. (Note that the commands would be typed on one line into the CLI.)

```
java weka.experiment.Experiment -r -T data/iris.arff
  -D weka.experiment.InstancesResultListener
  -P weka.experiment.RandomSplitResultProducer --
  -W weka.experiment.ClassifierSplitEvaluator --
  -W weka.classifiers.rules.OneR
```

While commands can be typed directly into the CLI, this technique is not particularly convenient and the experiments are not easy to modify.

The Experimenter comes in two flavours, either with a simple interface that provides most of the functionality one needs for experiments, or with an interface with full access to the Experimenter's capabilities. You can choose between those two with the *Experiment Configuration Mode* radio buttons:

- Simple
- Advanced

Both setups allow you to setup *standard* experiments, that are run locally on a single machine, or remote experiments, which are distributed between several hosts. The distribution of experiments cuts down the time the experiments will take until completion, but on the other hand the setup takes more time.

The next section covers the *standard* experiments (both, simple and advanced), followed by the *remote* experiments and finally the *analysing* of the results.

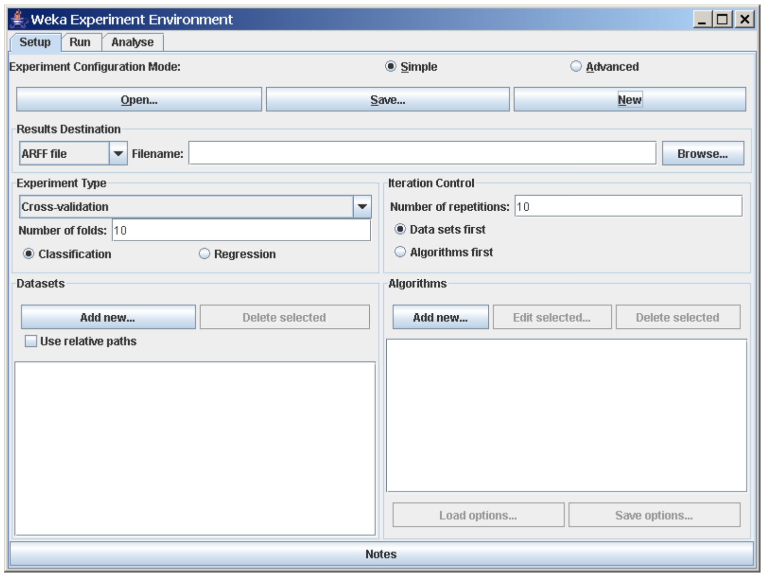
This manual is also available online on the *WekaDoc Wiki* [5].

2 Standard Experiments

Simple

New experiment

After clicking *New* default parameters for an Experiment are defined.



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

Algorithms

Results destination

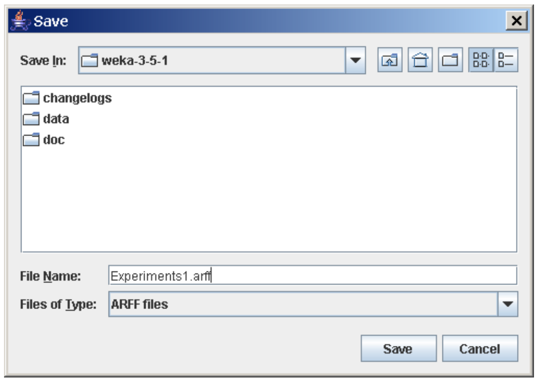
By default, an ARFF file is the destination for the results output. But you can choose between

- ARFF file
- CSV file
- JDBC database

ARFF file and JDBC database are discussed in detail in the following sections. CSV is similar to ARFF, but it can be used to be loaded in an external spreadsheet application.

ARFF file

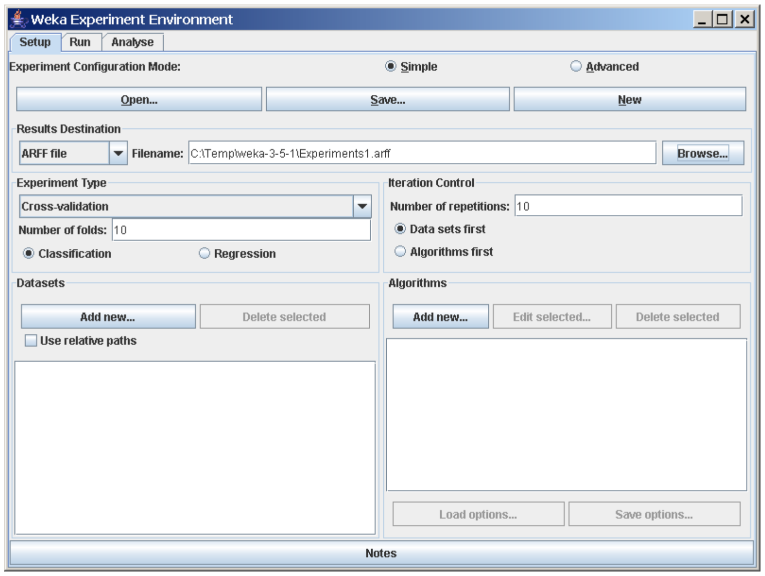
If the file name is left empty a temporary file will be created in the TEMP directory of the system. If one wants to specify an explicit results file, click on *Browse* and choose a filename, e.g., *Experiment1.arff*.



File Name: Experiments1.arff

Files of Type: ARFF files

Click on *Save* and the name will appear in the edit field next to *ARFF file*.



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: ARFF file Filename: C:\Temp\weka-3-5-1\Experiments1.arff

Experiment Type: Cross-validation

Number of folds: 10

[✓ Classification] Regression

Iteration Control

Number of repetitions: 10

[✓ Data sets first] Algorithms first

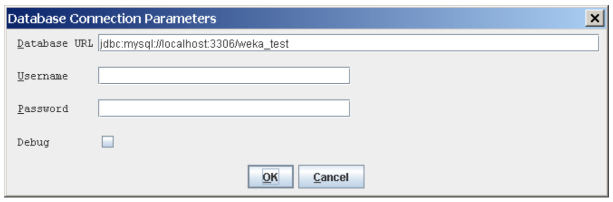
Use relative paths

The advantage of ARFF or CSV files is that they can be created without any additional classes besides the ones from Weka. The drawback is the lack of the ability to resume an experiment that was interrupted, e.g., due to an error or the addition of dataset or algorithms. Especially with time-consuming experiments, this behavior can be annoying.

JDBC database

With JDBC it is easy to store the results in a database. The necessary jar archives have to be in the CLASSPATH to make the JDBC functionality of a particular database available.

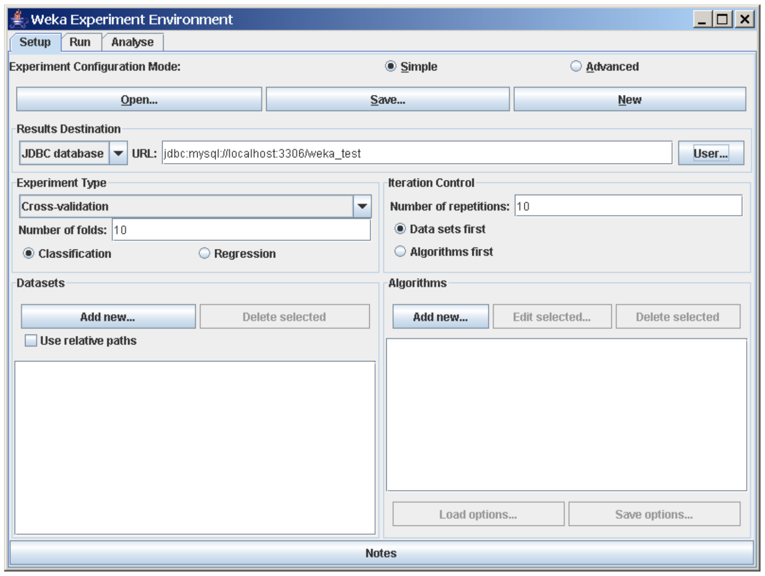
After changing *ARFF file* to *JDBC database* click on *User...* to specify JDBC URL and user credentials for accessing the database.



Database URL: jdbc:mysql://localhost:3306/weka_test
Username:
Password:
Debug: ☐

After supplying the necessary data and clicking on *OK*, the URL in the main window will be updated.

Note: at this point, the database connection is not tested; this is done when the experiment is started.



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: JDBC database URL: jdbc:mysql://localhost:3306/weka_test

Experiment Type: Cross-validation
Number of folds: 10
[✓ Classification] Regression

Iteration Control
Number of repetitions: 10
[✓ Data sets first] Algorithms first

Datasets
☐ Use relative paths

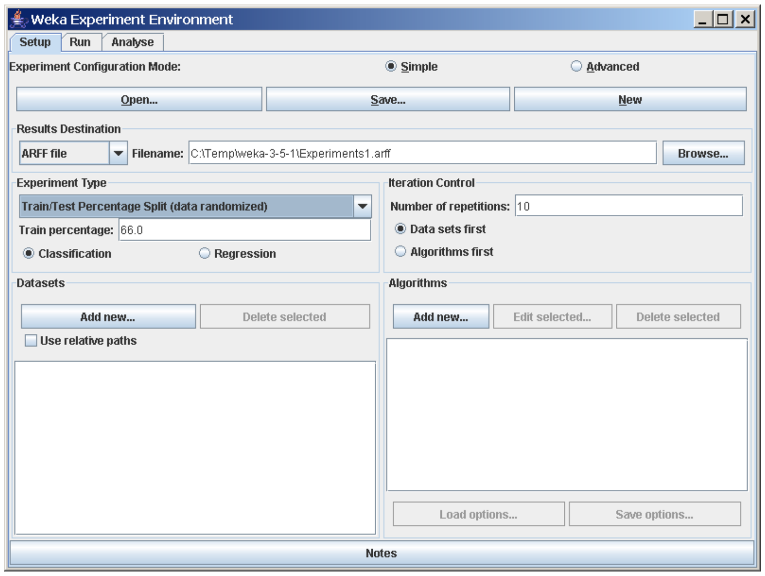
Algorithms

The advantage of a JDBC database is the possibility to resume an interrupted or extended experiment. Instead of re-running all the other algorithm/dataset combinations again, only the missing ones are computed.

Experiment type

The user can choose between the following three different types

- **Cross-validation (default)**
  performs stratified cross-validation with the given number of folds
- **Train/Test Percentage Split (data randomized)**
  splits a dataset according to the given percentage into a train and a test file (one cannot specify explicit training and test files in the Experimenter), after the order of the data has been randomized and stratified



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: ARFF file — Filename: C:\Temp\weka-3-5-1\Experiments1.arff

Experiment Type: Train/Test Percentage Split (data randomized)

Train percentage: 66.0

[✓ Classification] Regression

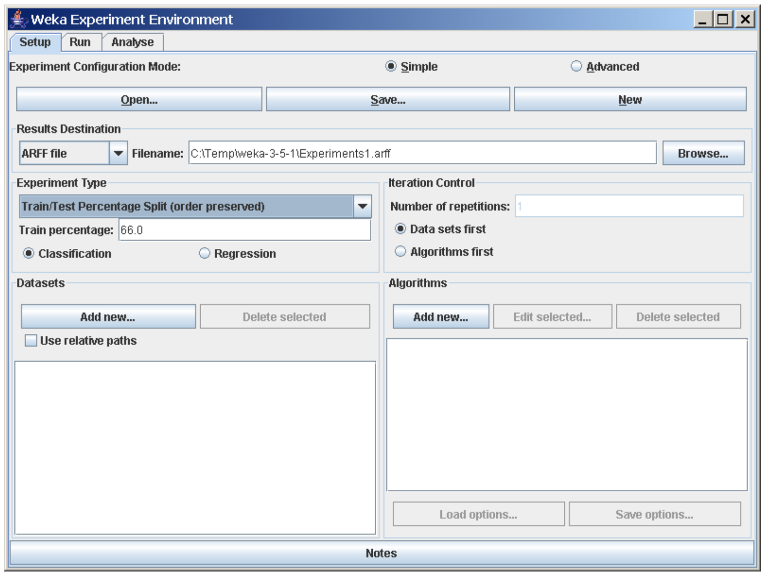
Iteration Control — Number of repetitions: 10

[✓ Data sets first] Algorithms first

Datasets — Use relative paths [ ]

Algorithms

- **Train/Test Percentage Split (order preserved)**
  because it is impossible to specify an explicit train/test files pair, one can *abuse* this type to *un-merge* previously merged train and test file into the two original files (one only needs to find out the correct percentage)



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: ARFF file — Filename: C:\Temp\weka-3-5-1\Experiments1.arff

Experiment Type: Train/Test Percentage Split (order preserved)

Train percentage: 66.0

[✓ Classification] Regression

Iteration Control — Number of repetitions: 1

[✓ Data sets first] Algorithms first

Datasets — Use relative paths [ ]

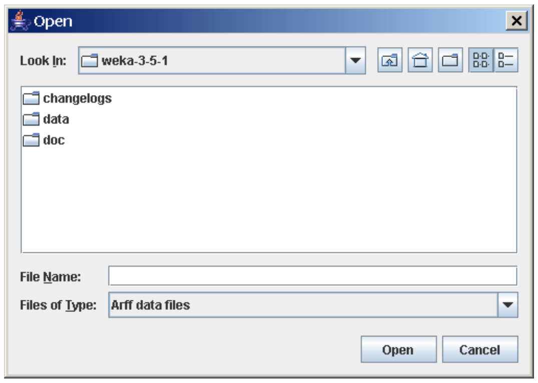
Algorithms

Additionally, one can choose between *Classification* and *Regression*, depending on the datasets and classifiers one uses. For decision trees like `J48` (Weka's implementation of Quinlan's C4.5 [3]) and the iris dataset, *Classification* is necessary, for a numeric classifier like `M5P`, on the other hand, *Regression*. *Classification* is selected by default.

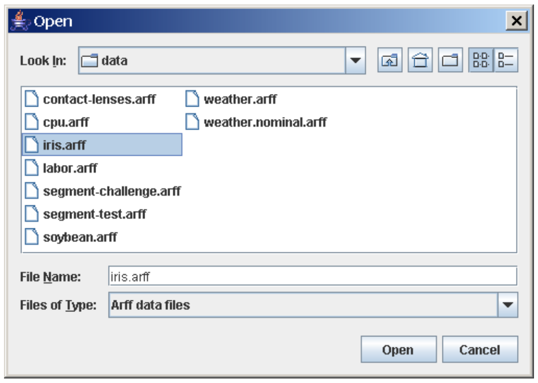
Note: if the percentage splits are used, one has to make sure that the corrected paired *T-Tester* still produces sensible results with the given ratio [2].

Datasets

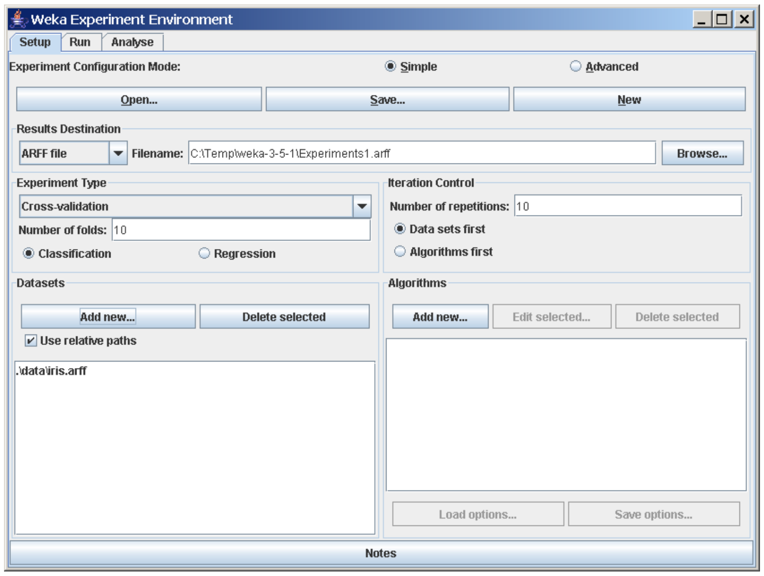
One can add dataset files either with an absolute path or with a relative one. The latter makes it often easier to run experiments on different machines, hence one should check *Use relative paths*, before clicking on *Add new...*.



In this example, open the *data* directory and choose the *iris.arff* dataset.



After clicking *Open* the file will be displayed in the datasets list. If one selects a directory and hits *Open*, then all ARFF files will be added recursively. Files can be deleted from the list by selecting them and then clicking on *Delete selected*.



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: ARFF file Filename: C:\Temp\weka-3-5-1\Experiments1.arff

Experiment Type: Cross-validation
Number of folds: 10
[✓ Classification] Regression

Iteration Control
Number of repetitions: 10
[✓ Data sets first] Algorithms first

Datasets
[✓ Use relative paths]
.\data\iris.arff

Algorithms

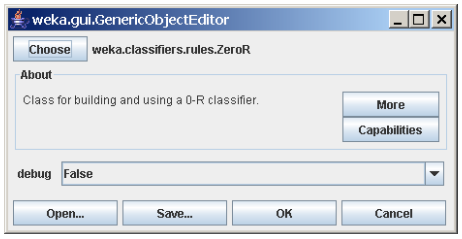
Iteration control

- **Number of repetitions**
  In order to get statistically meaningful results, the default number of iterations is 10. In case of 10-fold cross-validation this means 100 calls of one classifier with training data and tested against test data.

- **Data sets first/Algorithms first**
  As soon as one has more than one dataset and algorithm, it can be useful to switch from datasets being iterated over first to algorithms. This is the case, if one stores the results in a database and wants to complete the results for all the datasets for one algorithm as early as possible.

Algorithms

New algorithms can be added via the *Add new...* button. Opening this dialog for the first time, `ZeroR` is presented, otherwise the one that was selected last.

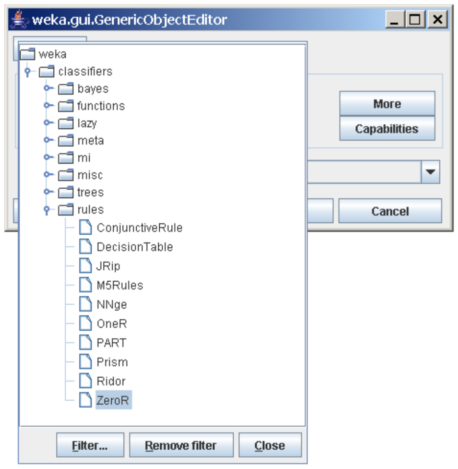


weka.classifiers.rules.ZeroR

About
Class for building and using a 0-R classifier.

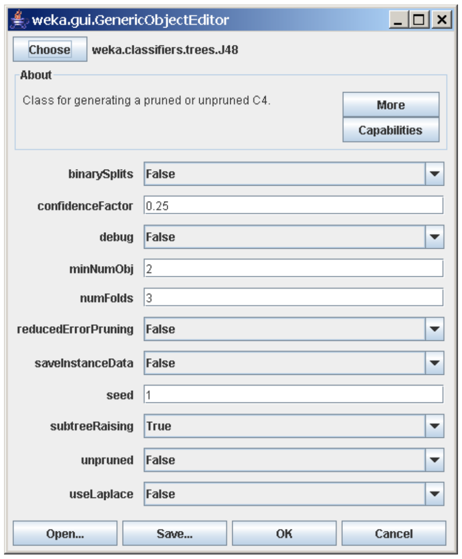
debug False

With the *Choose* button one can open the *GenericObjectEditor* and choose another classifier.



The *Filter...* button enables one to highlight classifiers that can handle certain attribute and class types. With *Remove filter* the highlighting will be removed again.

Additional algorithms can be added again with the *Add new...* button, e.g., the `J48` decision tree.



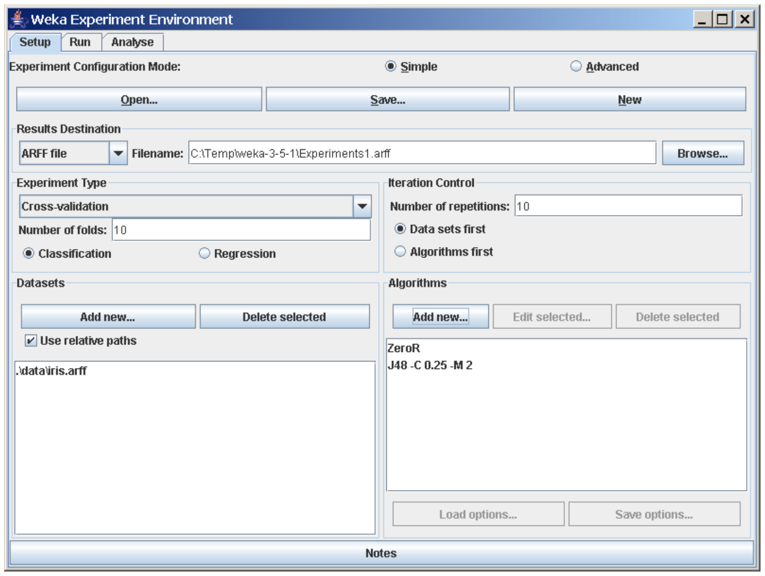
weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

binarySplits	False
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False

After setting the classifier parameters, one clicks on *OK* to add it to the list of algorithms.



Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: ARFF file — Filename: C:\Temp\weka-3-5-1\Experiments1.arff

Experiment Type: Cross-validation — Number of folds: 10 — [✓ Classification] Regression

Iteration Control: Number of repetitions: 10 — [✓ Data sets first] Algorithms first

Datasets: [✓ Use relative paths]

.\data\iris.arff

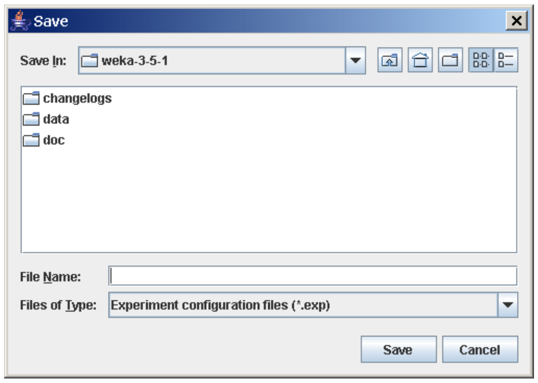
Algorithms:

ZeroR
J48 -C 0.25 -M 2

With the *Load options...* and *Save options...* buttons one can load and save the setup of a selected classifier from and to XML. This is especially useful for highly configured classifiers (e.g., nested meta-classifiers), where the manual setup takes quite some time, and which are used often.

Saving the setup

For future re-use, one can save the current setup of the experiment to a file by clicking on *Save...* at the top of the window.



Save In: weka-3-5-1

changelogs
data
doc

File Name:

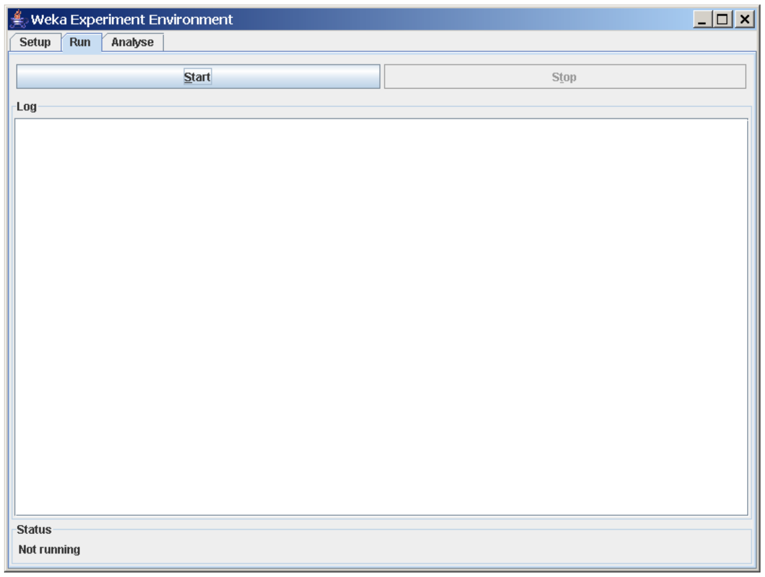
Files of Type: Experiment configuration files (*.exp)

By default, the format of the experiment files is the binary format that Java serialization offers. The drawback of this format is the possible incompatibility between different versions of Weka. A more robust alternative to the binary format is the XML format.

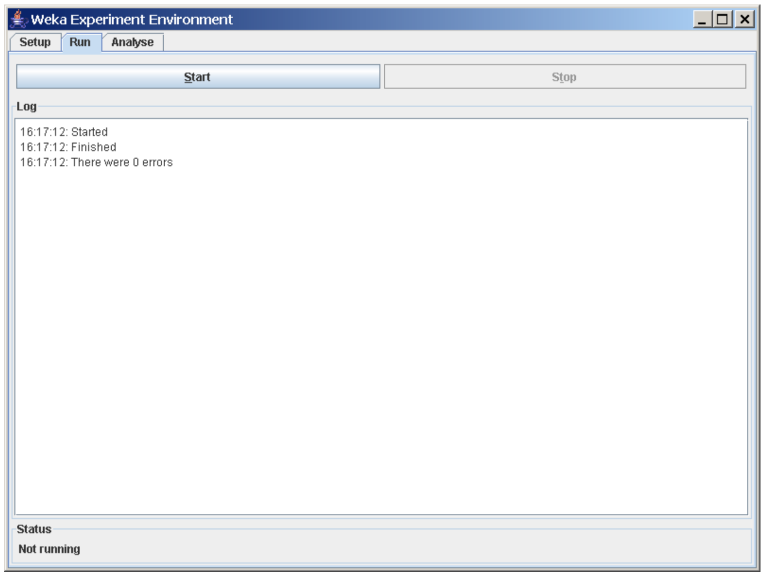
Previously saved experiments can be loaded again via the *Open...* button.

Running an Experiment

To run the current experiment, click the *Run* tab at the top of the Experiment Environment window. The current experiment performs 10 runs of 10-fold stratified cross-validation on the Iris dataset using the `ZeroR` and `J48` scheme.



Click *Start* to run the experiment.



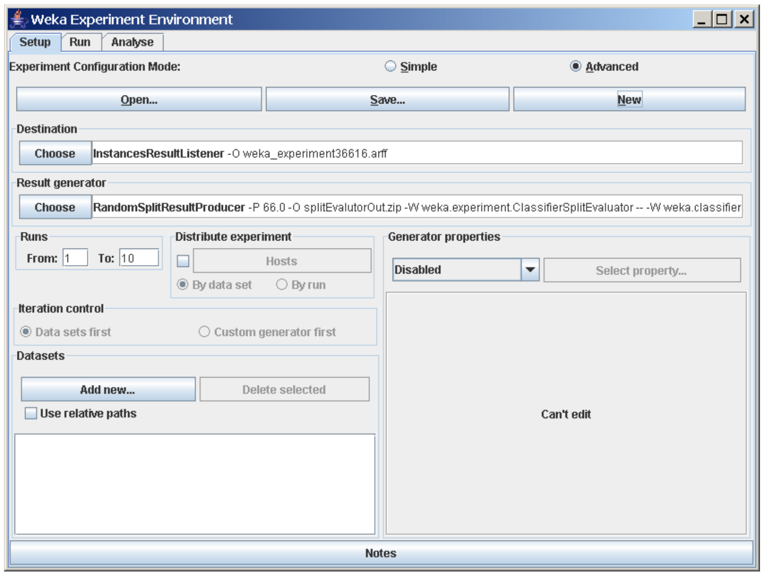
```
16:17:12: Started
16:17:12: Finished
16:17:12: There were 0 errors
```

If the experiment was defined correctly, the 3 messages shown above will be displayed in the *Log* panel. The results of the experiment are saved to the dataset *Experiment1.arff*.

Advanced

Defining an Experiment

When the Experimenter is started in *Advanced* mode, the *Setup* tab is displayed. Click *New* to initialize an experiment. This causes default parameters to be defined for the experiment.



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O weka_experiment36616.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs From: 1 To: 10

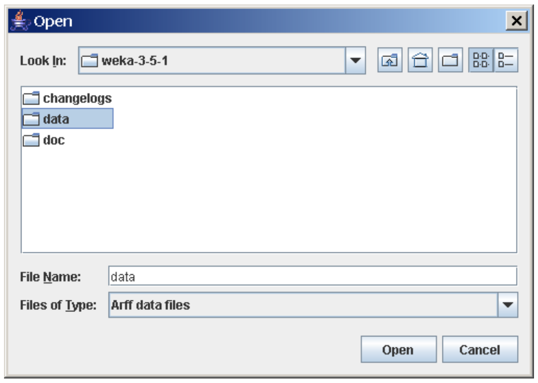
Distribute experiment: [✓ By data set] By run

Generator properties: Disabled

Iteration control: [✓ Data sets first] Custom generator first

Datasets: Use relative paths

To define the dataset to be processed by a scheme, first select *Use relative paths* in the *Datasets* panel of the *Setup* tab and then click on *Add new...* to open a dialog window.



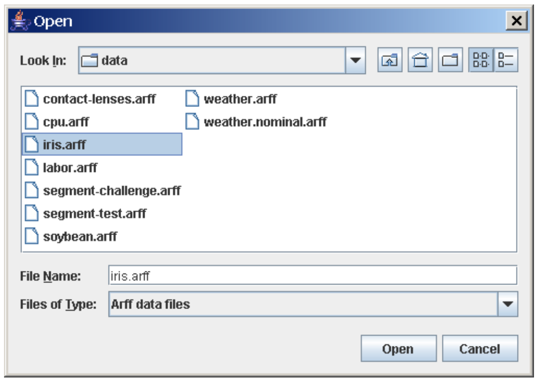
Look In: weka-3-5-1

changelogs
data
doc

File Name: data

Files of Type: Arff data files

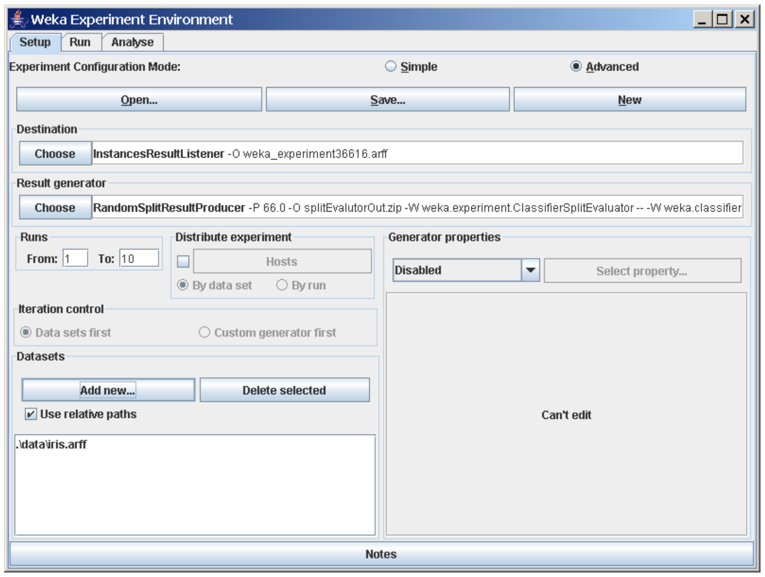
Double click on the *data* folder to view the available datasets or navigate to an alternate location. Select *iris.arff* and click *Open* to select the Iris dataset.



Look In: data

contact-lenses.arff
cpu.arff
iris.arff
labor.arff
segment-challenge.arff
segment-test.arff
soybean.arff
weather.arff
weather.nominal.arff

File Name: iris.arff
Files of Type: Arff data files



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O weka_experiment36616.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs From: 1 To: 10

Distribute experiment: [✓ By data set] By run

Generator properties: Disabled

Iteration control: [✓ Data sets first] Custom generator first

Datasets

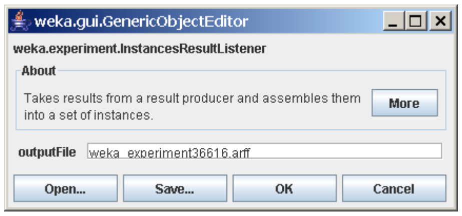
✓ Use relative paths

.\data\iris.arff

The dataset name is now displayed in the *Datasets* panel of the *Setup* tab.

Saving the Results of the Experiment

To identify a dataset to which the results are to be sent, click on the *Instances-ResultListener* entry in the *Destination* panel. The output file parameter is near the bottom of the window, beside the text *outputFile*. Click on this parameter to display a file selection window.

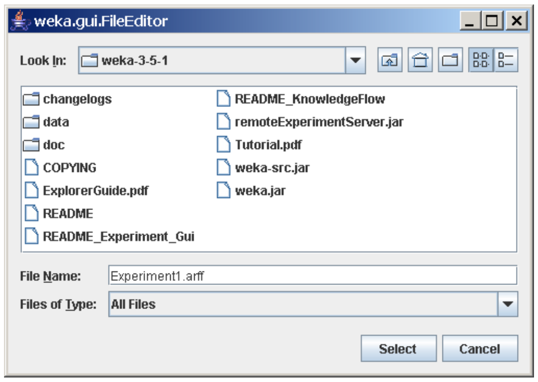


weka.experiment.InstancesResultListener

About

Takes results from a result producer and assembles them into a set of instances.

outputFile weka_experiment36616.arff

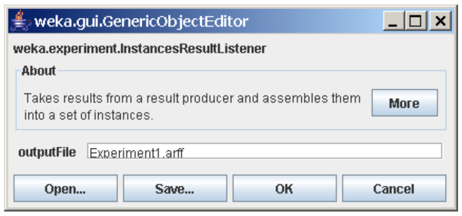

weka.gui.FileEditor

Look In: weka-3-5-1

changelogs
data
doc
COPYING
ExplorerGuide.pdf
README
README_Experiment_Gui
README_KnowledgeFlow
remoteExperimentServer.jar
Tutorial.pdf
weka-src.jar
weka.jar

File Name: Experiment1.arff
Files of Type: All Files

Type the name of the output file, click *Select*, and then click close (x). The file name is displayed in the *outputFile* panel. Click on *OK* to close the window.

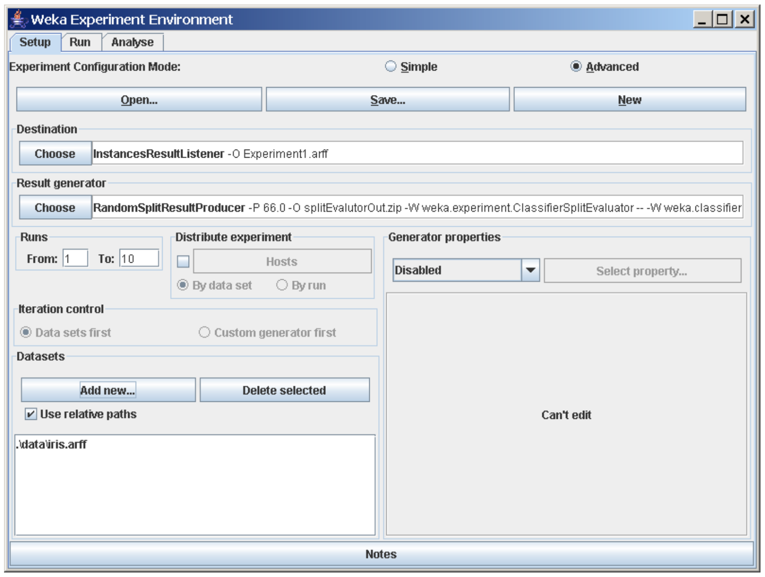

weka.gui.GenericObjectEditor

weka.experiment.InstancesResultListener

About
Takes results from a result producer and assembles them into a set of instances.

outputFile Experiment1.arff

The dataset name is displayed in the *Destination* panel of the *Setup* tab.


Weka Experiment Environment

Experiment Configuration Mode: ○ Simple [✓ ● Advanced]

Destination
InstancesResultListener -O Experiment1.arff

Result generator
RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs
From: 1 To: 10

Distribute experiment
[✓ ● By data set] ○ By run

Generator properties
Disabled

Iteration control
[✓ ● Data sets first] ○ Custom generator first

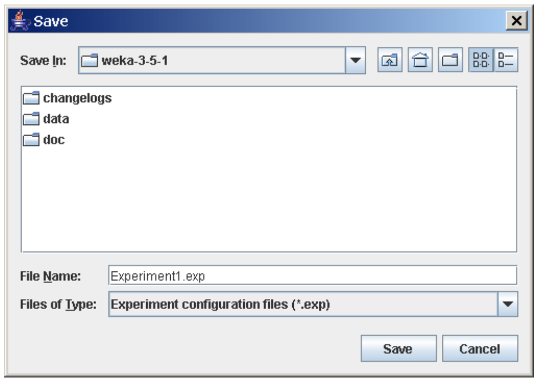
Datasets
[✓ Use relative paths]

.\data\iris.arff

Can't edit

Saving the Experiment Definition

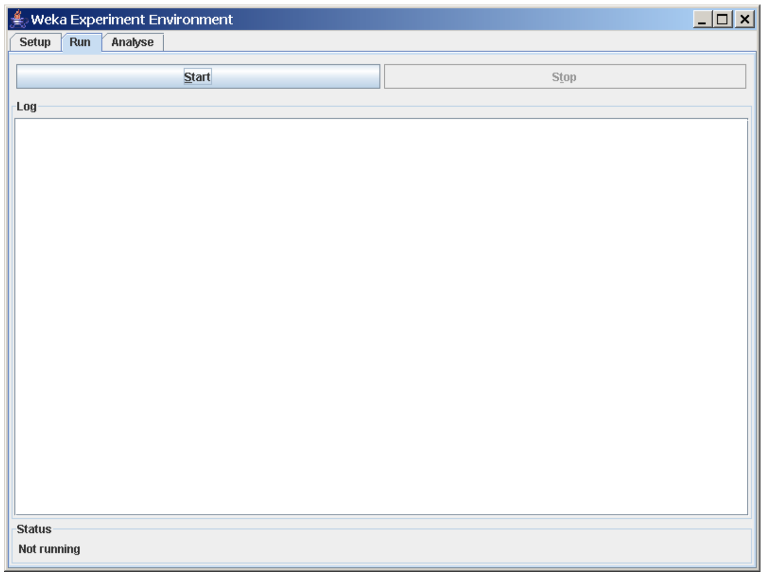
The experiment definition can be saved at any time. Select *Save...* at the top of the *Setup* tab. Type the dataset name with the extension *exp* (or select the dataset name if the experiment definition dataset already exists) for binary files or choose *Experiment configuration files (*.xml)* from the file types combobox (the XML files are robust with respect to version changes).



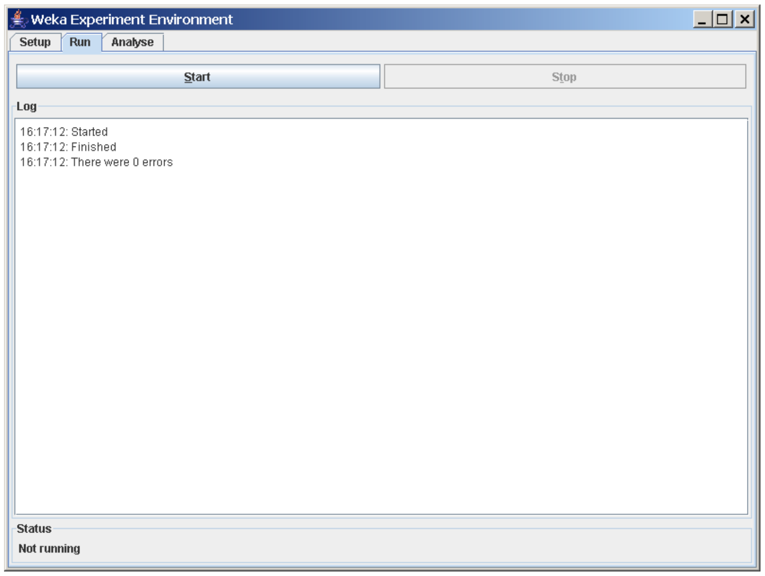
The experiment can be restored by selecting *Open* in the *Setup* tab and then selecting *Experiment1.exp* in the dialog window.

Running an Experiment

To run the current experiment, click the *Run* tab at the top of the Experiment Environment window. The current experiment performs 10 randomized train and test runs on the Iris dataset, using 66% of the patterns for training and 34% for testing, and using the `ZeroR` scheme.



Click *Start* to run the experiment.



```
16:17:12: Started
16:17:12: Finished
16:17:12: There were 0 errors
```

```
Not running
```

If the experiment was defined correctly, the 3 messages shown above will be displayed in the *Log* panel. The results of the experiment are saved to the dataset *Experiment1.arff*. The first few lines in this dataset are shown below.

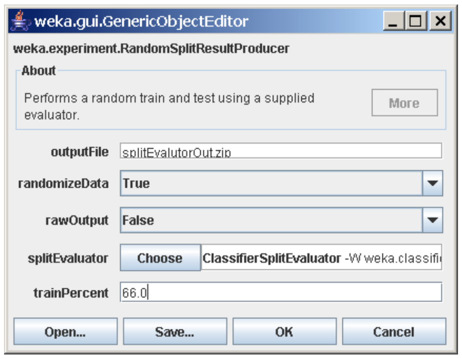
```
@relation InstanceResultListener

@attribute Key_Dataset {iris}
@attribute Key_Run {1,2,3,4,5,6,7,8,9,10}
@attribute Key_Scheme {weka.classifiers.rules.ZeroR,weka.classifiers.trees.J48}
@attribute Key_Scheme_options {,'-C 0.25 -M 2'}
@attribute Key_Scheme_version_ID {48055541465867954,-217733168393644444}
@attribute Date_time numeric
@attribute Number_of_training_instances numeric
@attribute Number_of_testing_instances numeric
@attribute Number_correct numeric
@attribute Number_incorrect numeric
@attribute Number_unclassified numeric
@attribute Percent_correct numeric
@attribute Percent_incorrect numeric
@attribute Percent_unclassified numeric
@attribute Kappa_statistic numeric
@attribute Mean_absolute_error numeric
@attribute Root_mean_squared_error numeric
@attribute Relative_absolute_error numeric
@attribute Root_relative_squared_error numeric
@attribute SF_prior_entropy numeric
@attribute SF_scheme_entropy numeric
@attribute SF_entropy_gain numeric
@attribute SF_mean_prior_entropy numeric
@attribute SF_mean_scheme_entropy numeric
@attribute SF_mean_entropy_gain numeric
@attribute KB_information numeric
```

```
@attribute KB_mean_information numeric
@attribute KB_relative_information numeric
@attribute True_positive_rate numeric
@attribute Num_true_positives numeric
@attribute False_positive_rate numeric
@attribute Num_false_positives numeric
@attribute True_negative_rate numeric
@attribute Num_true_negatives numeric
@attribute False_negative_rate numeric
@attribute Num_false_negatives numeric
@attribute IR_precision numeric
@attribute IR_recall numeric
@attribute F_measure numeric
@attribute Area_under_ROC numeric
@attribute Time_training numeric
@attribute Time_testing numeric
@attribute Summary {'Number of leaves: 3\nSize of the tree: 5\n',
   'Number of leaves: 5\nSize of the tree: 9\n',
   'Number of leaves: 4\nSize of the tree: 7\n'}
@attribute measureTreeSize numeric
@attribute measureNumLeaves numeric
@attribute measureNumRules numeric

@data

iris,1,weka.classifiers.rules.ZeroR,,48055541465867954,20051221.033,99,51,
17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,
0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0,0,?,?,?,?
```

Changing the Experiment Parameters

Changing the Classifier

The parameters of an experiment can be changed by clicking on the *Result generator* panel.


weka.gui.GenericObjectEditor

weka.experiment.RandomSplitResultProducer

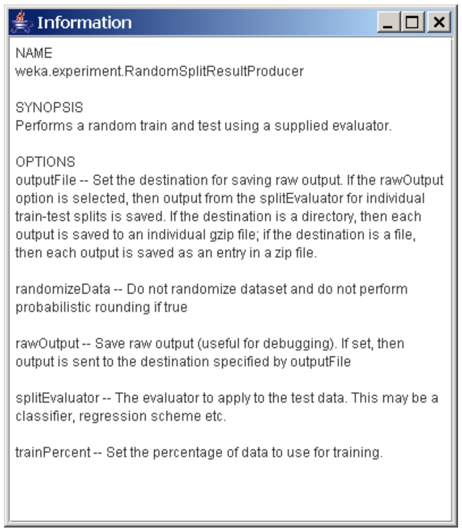
About

Performs a random train and test using a supplied evaluator.

outputFile: splitEvalutorOut.zip
randomizeData: True
rawOutput: False
splitEvaluator: ClassifierSplitEvaluator -W weka.classifi
trainPercent: 66.0

The *RandomSplitResultProducer* performs repeated train/test runs. The number of instances (expressed as a percentage) used for training is given in the *trainPercent* box. (The number of runs is specified in the *Runs* panel in the *Setup* tab.)

A small help file can be displayed by clicking *More* in the *About* panel.


Information

NAME
weka.experiment.RandomSplitResultProducer

SYNOPSIS
Performs a random train and test using a supplied evaluator.

OPTIONS
outputFile -- Set the destination for saving raw output. If the rawOutput option is selected, then output from the splitEvaluator for individual train-test splits is saved. If the destination is a directory, then each output is saved to an individual gzip file; if the destination is a file, then each output is saved as an entry in a zip file.

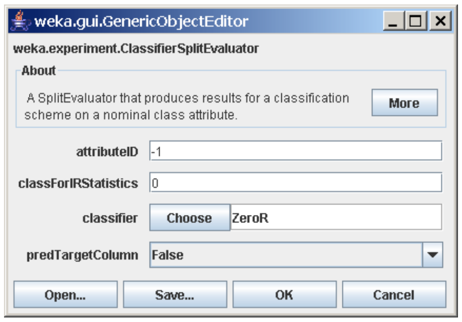
randomizeData -- Do not randomize dataset and do not perform probabilistic rounding if true

rawOutput -- Save raw output (useful for debugging). If set, then output is sent to the destination specified by outputFile

splitEvaluator -- The evaluator to apply to the test data. This may be a classifier, regression scheme etc.

trainPercent -- Set the percentage of data to use for training.

Click on the *splitEvaluator* entry to display the *SplitEvaluator* properties.


weka.gui.GenericObjectEditor

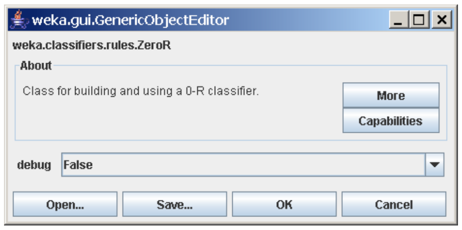
weka.experiment.ClassifierSplitEvaluator

About

A SplitEvaluator that produces results for a classification scheme on a nominal class attribute.

attributeID: -1
classForIRStatistics: 0
classifier: ZeroR
predTargetColumn: False

Click on the classifier entry (`ZeroR`) to display the scheme properties.


weka.gui.GenericObjectEditor

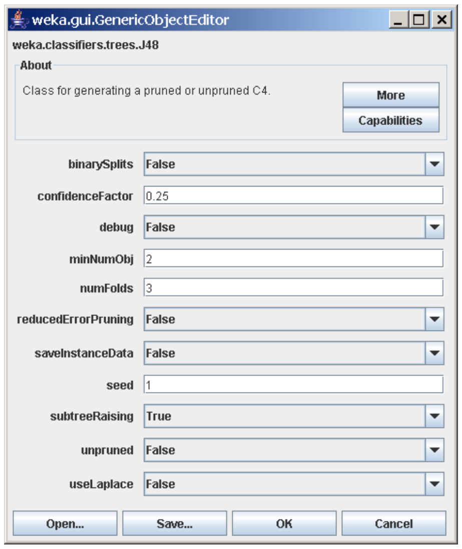
weka.classifiers.rules.ZeroR

About

Class for building and using a 0-R classifier.

debug: False

This scheme has no modifiable properties (besides *debug* mode on/off) but most other schemes do have properties that can be modified by the user. The *Capabilities* button opens a small dialog listing all the attribute and class types this classifier can handle. Click on the *Choose* button to select a different scheme. The window below shows the parameters available for the `J48` decision-tree scheme. If desired, modify the parameters and then click *OK* to close the window.


weka.gui.GenericObjectEditor

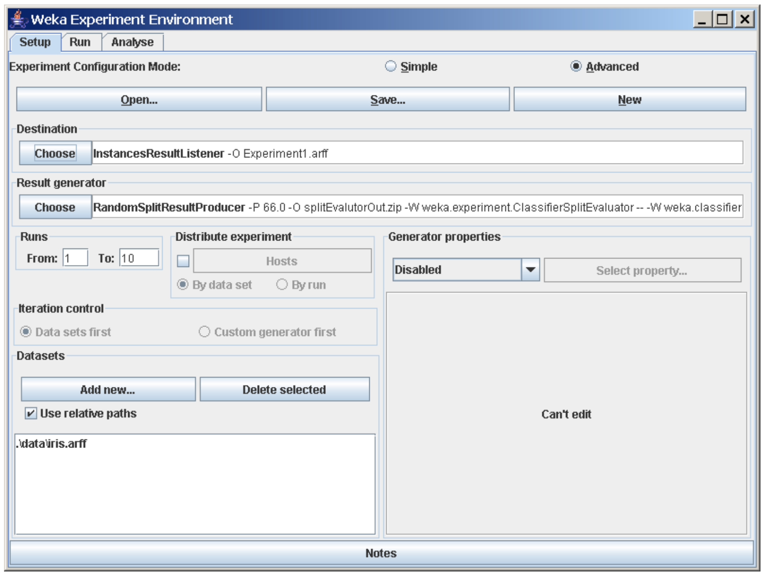
weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

binarySplits: False
confidenceFactor: 0.25
debug: False
minNumObj: 2
numFolds: 3
reducedErrorPruning: False
saveInstanceData: False
seed: 1
subtreeRaising: True
unpruned: False
useLaplace: False

The name of the new scheme is displayed in the *Result generator* panel.


Weka Experiment Environment

Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs: From: 1 To: 10

Distribute experiment: [✓ By data set] By run

Generator properties: Disabled

Iteration control: [✓ Data sets first] Custom generator first

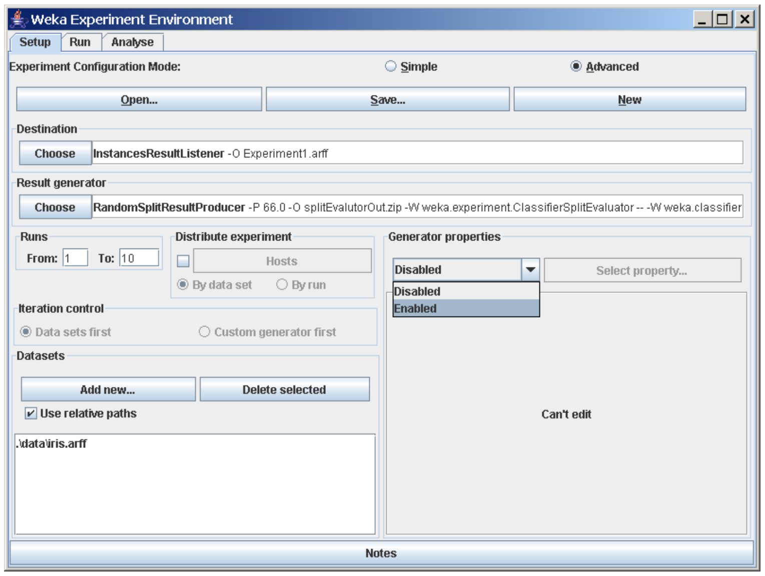
Datasets: [✓ Use relative paths]

.\data\iris.arff

Can't edit

Adding Additional Schemes

Additional schemes can be added in the *Generator properties* panel. To begin, change the drop-down list entry from *Disabled* to *Enabled* in the *Generator properties* panel.



Experiment Configuration Mode: ○ Simple ● Advanced

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs From: 1 To: 10

Distribute experiment: ☐ ● By data set ○ By run

Generator properties: Disabled (Disabled / Enabled)

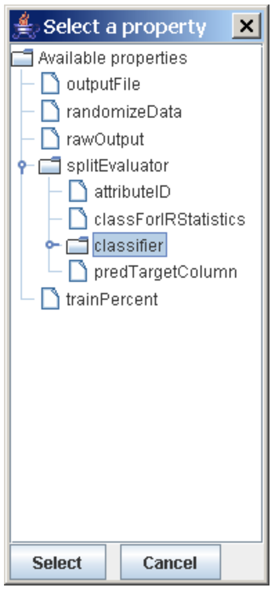
Iteration control: ● Data sets first ○ Custom generator first

Datasets: ☑ Use relative paths

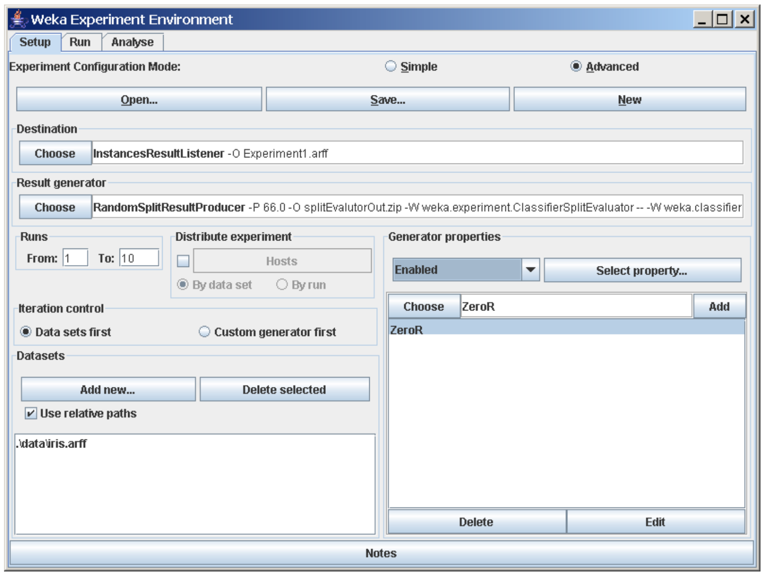
.\data\iris.arff

Can't edit

Click *Select property* and expand *splitEvaluator* so that the *classifier* entry is visible in the property list; click *Select*.



The scheme name is displayed in the *Generator properties* panel.



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs: From: 1 To: 10

Distribute experiment: [ ] [✓ By data set] By run

Generator properties: Enabled

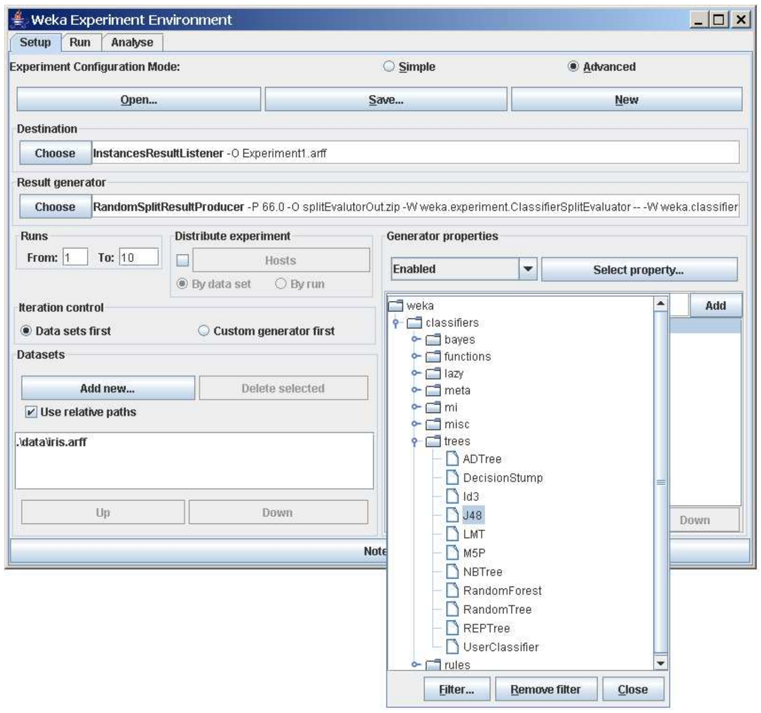
Iteration control: [✓ Data sets first] Custom generator first

Datasets: [✓ Use relative paths]

.\data\iris.arff

ZeroR

To add another scheme, click on the *Choose* button to display the *Generic-ObjectEditor* window.



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs: From: 1 To: 10

Distribute experiment: [ ] [✓ By data set] By run

Generator properties: Enabled

Iteration control: [✓ Data sets first] Custom generator first

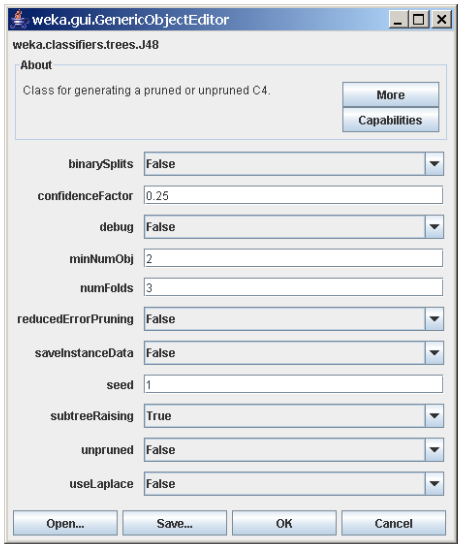
Datasets: [✓ Use relative paths]

.\data\iris.arff

```
weka
  classifiers
    bayes
    functions
    lazy
    meta
    mi
    misc
    trees
      ADTree
      DecisionStump
      Id3
      J48
      LMT
      M5P
      NBTree
      RandomForest
      RandomTree
      REPTree
      UserClassifier
    rules
```

The *Filter...* button enables one to highlight classifiers that can handle certain attribute and class types. With *Remove filter* the highlighting will be removed again.

To change to a decision-tree scheme, select `J48` (in subgroup *trees*).



weka.gui.GenericObjectEditor

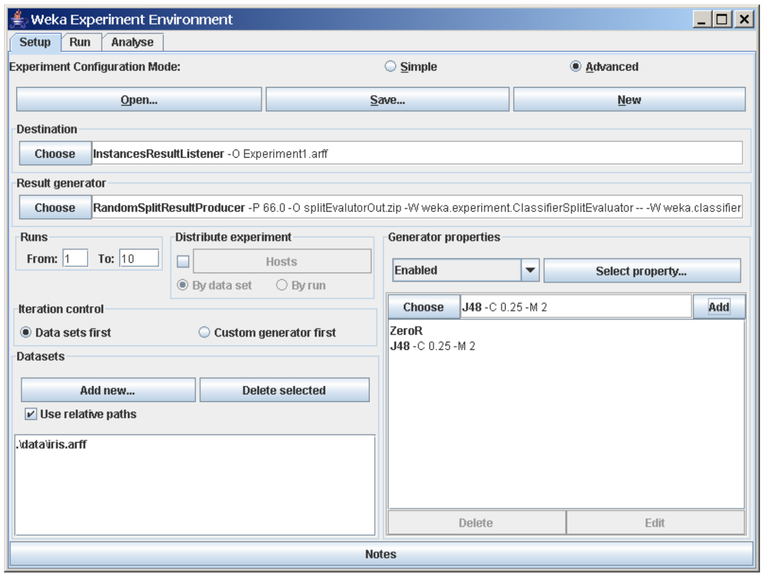
weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

binarySplits	False
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False

The new scheme is added to the *Generator properties* panel. Click *Add* to add the new scheme.



Weka Experiment Environment

Experiment Configuration Mode: ( ) Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs: From: 1 To: 10

Distribute experiment: Hosts; [✓ By data set] ( ) By run

Generator properties: Enabled

J48 -C 0.25 -M 2

```
ZeroR
J48 -C 0.25 -M 2
```

Iteration control: [✓ Data sets first] ( ) Custom generator first

Datasets: [✓] Use relative paths

```
.\data\iris.arff
```

Now when the experiment is run, results are generated for both schemes.

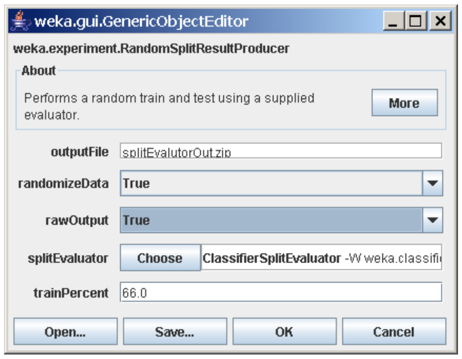
To add additional schemes, repeat this process. To remove a scheme, select the scheme by clicking on it and then click *Delete*.

Adding Additional Datasets

The scheme(s) may be run on any number of datasets at a time. Additional datasets are added by clicking *Add new...* in the *Datasets* panel. Datasets are deleted from the experiment by selecting the dataset and then clicking *Delete Selected*.

Raw Output

The raw output generated by a scheme during an experiment can be saved to a file and then examined at a later time. Open the *ResultProducer* window by clicking on the *Result generator* panel in the *Setup* tab.


weka.gui.GenericObjectEditor

weka.experiment.RandomSplitResultProducer

About

Performs a random train and test using a supplied evaluator.

outputFile: splitEvalutorOut.zip

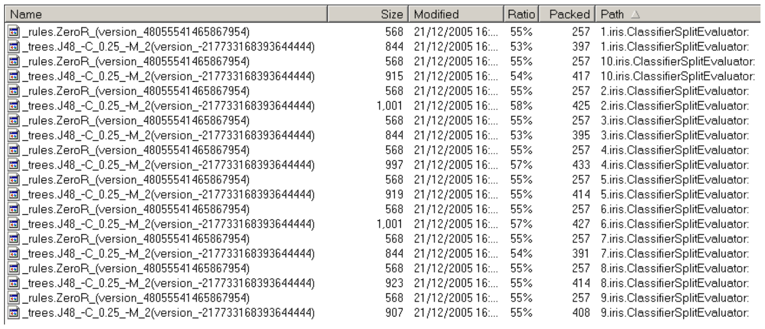
randomizeData: True

rawOutput: True

splitEvaluator: ClassifierSplitEvaluator -W weka.classifi

trainPercent: 66.0

Click on *rawOutput* and select the *True* entry from the drop-down list. By default, the output is sent to the zip file *splitEvaluatorOut.zip*. The output file can be changed by clicking on the *outputFile* panel in the window. Now when the experiment is run, the result of each processing run is archived, as shown below.



Name	Size	Modified	Ratio	Packed	Path
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	1.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	844	21/12/2005 16:...	53%	397	1.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	10.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	915	21/12/2005 16:...	54%	417	10.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	2.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	1,001	21/12/2005 16:...	58%	425	2.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	3.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	844	21/12/2005 16:...	53%	395	3.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	4.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	997	21/12/2005 16:...	57%	433	4.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	5.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	919	21/12/2005 16:...	55%	414	5.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	6.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	1,001	21/12/2005 16:...	57%	427	6.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	7.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	844	21/12/2005 16:...	54%	391	7.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	8.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	923	21/12/2005 16:...	55%	414	8.iris.ClassifierSplitEvaluator:
rules.ZeroR(version_48055541465867954)	568	21/12/2005 16:...	55%	257	9.iris.ClassifierSplitEvaluator:
trees.J48-C_0.25_-M_2(version_-217733168393644444)	907	21/12/2005 16:...	55%	408	9.iris.ClassifierSplitEvaluator:

The contents of the first run are:

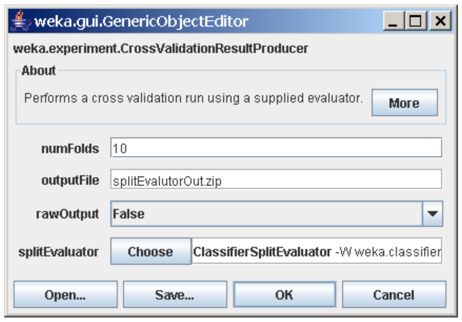
```
ClassifierSplitEvaluator: weka.classifiers.trees.J48 -C 0.25 -M 2(version
     -217733168393644444)Classifier model:
J48 pruned tree
------------------

petalwidth <= 0.6: Iris-setosa (33.0)
petalwidth > 0.6
|   petalwidth <= 1.5: Iris-versicolor (31.0/1.0)
|   petalwidth > 1.5: Iris-virginica (35.0/3.0)

Number of Leaves  :   3

Size of the tree :   5
```

```
Correctly Classified Instances          47               92.1569 %
Incorrectly Classified Instances         4                7.8431 %
Kappa statistic                          0.8824
Mean absolute error                      0.0723
Root mean squared error                  0.2191
Relative absolute error                 16.2754 %
Root relative squared error             46.4676 %
Total Number of Instances               51
measureTreeSize : 5.0
measureNumLeaves : 3.0
measureNumRules : 3.0
```

Other Result Producers

Cross-Validation Result Producer

To change from random train and test experiments to cross-validation experiments, click on the *Result generator* entry. At the top of the window, click on the drop-down list and select *CrossValidationResultProducer*. The window now contains parameters specific to cross-validation such as the number of partitions/folds. The experiment performs 10-fold cross-validation instead of train and test in the given example.


weka.gui.GenericObjectEditor

weka.experiment.CrossValidationResultProducer

About

Performs a cross validation run using a supplied evaluator.

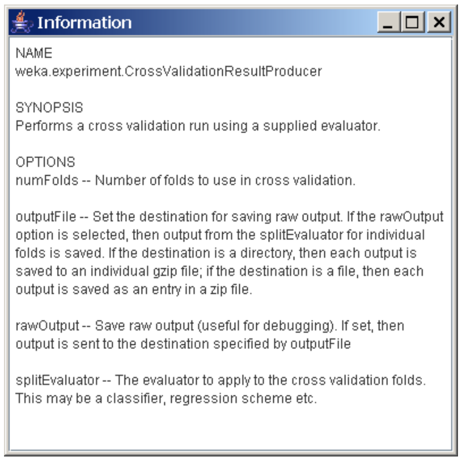
numFolds: 10

outputFile: splitEvalutorOut.zip

rawOutput: False

splitEvaluator: ClassifierSplitEvaluator -W weka.classifier

The *Result generator* panel now indicates that cross-validation will be performed. Click on *More* to generate a brief description of the *CrossValidationResultProducer*.


Information

NAME
weka.experiment.CrossValidationResultProducer

SYNOPSIS
Performs a cross validation run using a supplied evaluator.

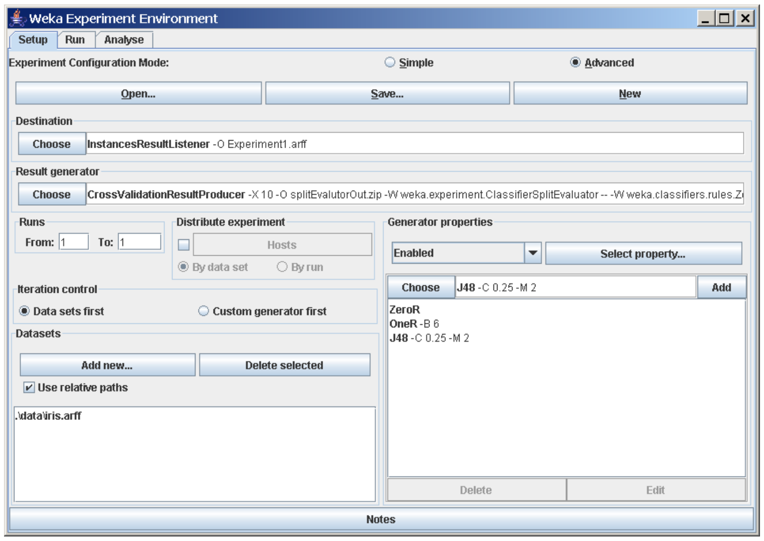
OPTIONS
numFolds -- Number of folds to use in cross validation.

outputFile -- Set the destination for saving raw output. If the rawOutput option is selected, then output from the splitEvaluator for individual folds is saved. If the destination is a directory, then each output is saved to an individual gzip file; if the destination is a file, then each output is saved as an entry in a zip file.

rawOutput -- Save raw output (useful for debugging). If set, then output is sent to the destination specified by outputFile

splitEvaluator -- The evaluator to apply to the cross validation folds. This may be a classifier, regression scheme etc.

As with the *RandomSplitResultProducer*, multiple schemes can be run during cross-validation by adding them to the *Generator properties* panel.



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: CrossValidationResultProducer -X 10 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifiers.rules.Z

Runs From: 1 To: 1

Iteration control: [✓ Data sets first] Custom generator first

Use relative paths [✓]

Datasets: .\data\iris.arff

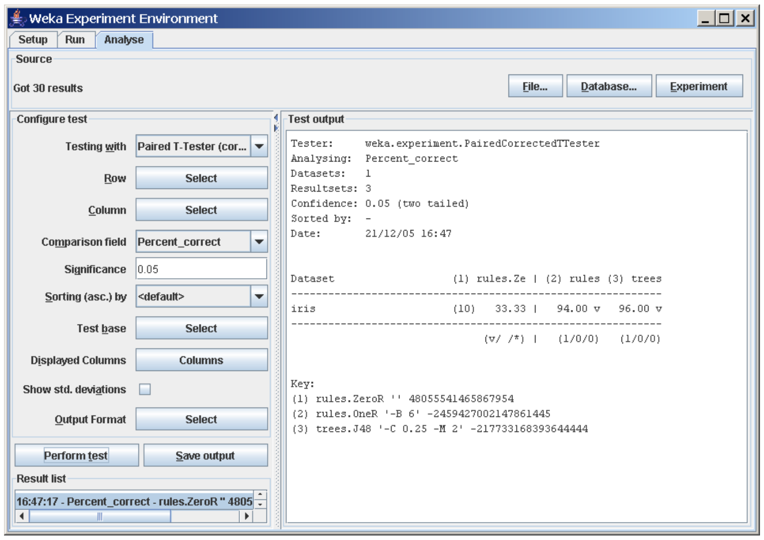
Generator properties: Enabled

J48 -C 0.25 -M 2

ZeroR
OneR -B 6
J48 -C 0.25 -M 2

The number of runs is set to 1 in the *Setup* tab in this example, so that only one run of cross-validation for each scheme and dataset is executed.

When this experiment is analysed, the following results are generated. Note that there are 30 (1 run times 10 folds times 3 schemes) result lines processed.



Source: Got 30 results

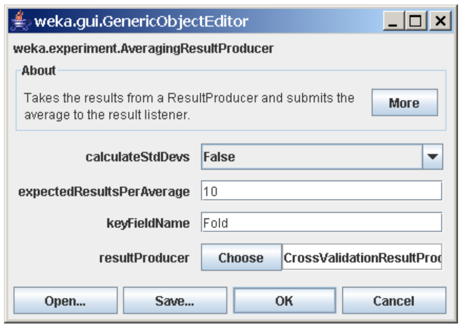
Testing with: Paired T-Tester (cor...
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>

Result list: 16:47:17 - Percent_correct - rules.ZeroR '' 4805

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:47


Dataset                   (1) rules.Ze | (2) rules (3) trees
------------------------------------------------------------
iris                      (10)   33.33 |   94.00 v   96.00 v
------------------------------------------------------------
                               (v/ /*) |   (1/0/0)   (1/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

Averaging Result Producer

An alternative to the *CrossValidationResultProducer* is the *AveragingResultProducer*. This result producer takes the average of a set of runs (which are typically cross-validation runs). This result producer is identified by clicking the *Result generator* panel and then choosing the *AveragingResultProducer* from the *GenericObjectEditor*.



weka.experiment.AveragingResultProducer

About

Takes the results from a ResultProducer and submits the average to the result listener.

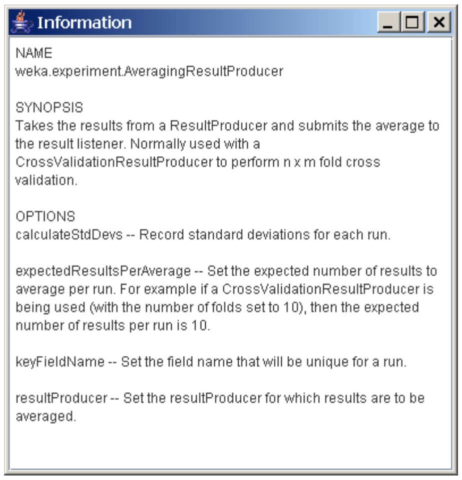
calculateStdDevs: False

expectedResultsPerAverage: 10

keyFieldName: Fold

resultProducer: Choose CrossValidationResultProd

The associated help file is shown below.



NAME
weka.experiment.AveragingResultProducer

SYNOPSIS
Takes the results from a ResultProducer and submits the average to the result listener. Normally used with a CrossValidationResultProducer to perform n x m fold cross validation.

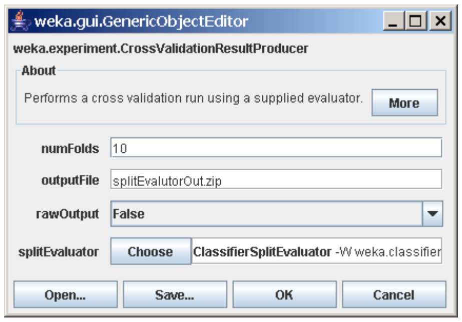
OPTIONS
calculateStdDevs -- Record standard deviations for each run.

expectedResultsPerAverage -- Set the expected number of results to average per run. For example if a CrossValidationResultProducer is being used (with the number of folds set to 10), then the expected number of results per run is 10.

keyFieldName -- Set the field name that will be unique for a run.

resultProducer -- Set the resultProducer for which results are to be averaged.

Clicking the *resultProducer* panel brings up the following window.



weka.experiment.CrossValidationResultProducer

About

Performs a cross validation run using a supplied evaluator.

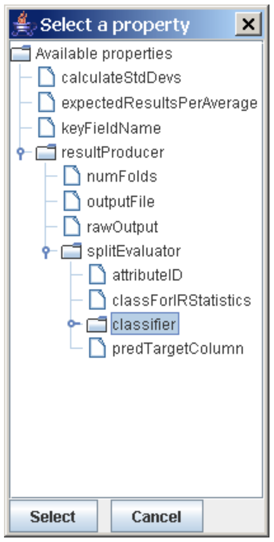
numFolds: 10

outputFile: splitEvalutorOut.zip

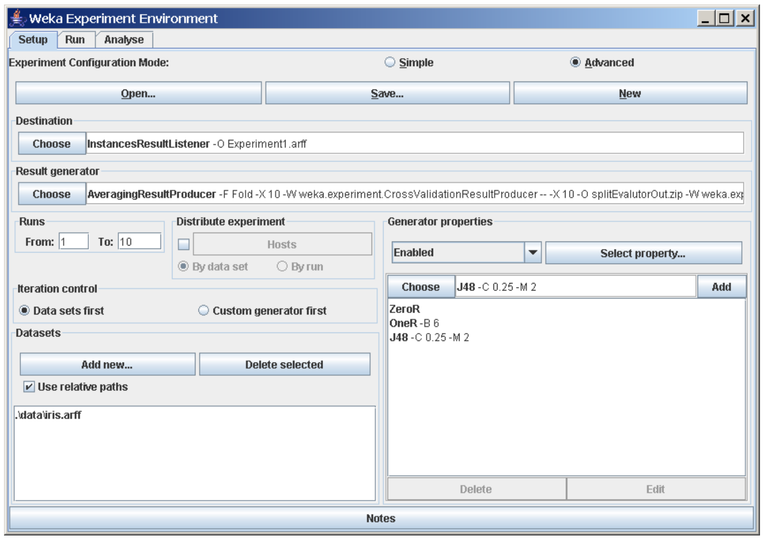
rawOutput: False

splitEvaluator: Choose ClassifierSplitEvaluator -W weka.classifier

As with the other ResultProducers, additional schemes can be defined. When the *AveragingResultProducer* is used, the classifier property is located deeper in the *Generator properties* hierarchy.



```
Available properties
  calculateStdDevs
  expectedResultsPerAverage
  keyFieldName
  resultProducer
    numFolds
    outputFile
    rawOutput
    splitEvaluator
      attributeID
      classForIRStatistics
      classifier
      predTargetColumn
```



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: AveragingResultProducer -F Fold -X 10 -W weka.experiment.CrossValidationResultProducer -- -X 10 -O splitEvalutorOut.zip -W weka.exp

Runs: From: 1 To: 10

Distribute experiment: [✓ By data set] By run

Iteration control: [✓ Data sets first] Custom generator first

Generator properties: Enabled

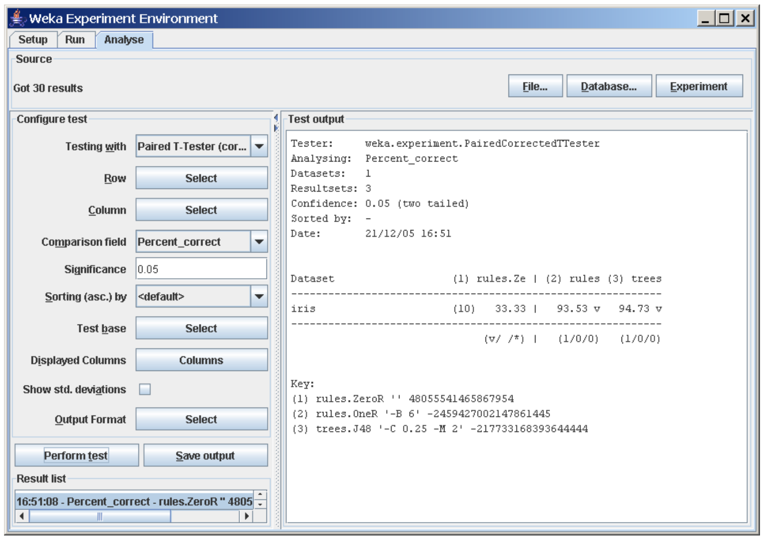
J48 -C 0.25 -M 2

ZeroR
OneR -B 6
J48 -C 0.25 -M 2

Datasets: ✓ Use relative paths

.\data\iris.arff

In this experiment, the `ZeroR`, `OneR`, and `J48` schemes are run 10 times with 10-fold cross-validation. Each set of 10 cross-validation folds is then averaged, producing one result line for each run (instead of one result line for each fold as in the previous example using the *CrossValidationResultProducer*) for a total of 30 result lines. If the raw output is saved, all 300 results are sent to the archive.



Got 30 results

Configure test

Testing with: Paired T-Tester (cor...

Comparison field: Percent_correct

Significance: 0.05

Sorting (asc.) by: <default>

Show std. deviations: ☐

Result list

16:51:08 - Percent_correct - rules.ZeroR '' 4805

Test output

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:51


Dataset                   (1) rules.Ze | (2) rules (3) trees
------------------------------------------------------------
iris                      (10)   33.33 |   93.53 v   94.73 v
------------------------------------------------------------
                               (v/ /*) |   (1/0/0)   (1/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

3 Remote Experiments

Remote experiments enable you to distribute the computing load across multiple computers.

Preparation

To run a remote experiment you will need:

- A database server
- A number of computers to run remote engines on
- To edit the remote engine policy file included in the Weka distribution to allow class loading from your home directory
- An invocation of the Experimenter on a machine somewhere (any will do)

For the following examples, we assume a user called *johndoe* with this setup:

- Access to a set of computers running a flavour of Unix (pathnames need to be changed for Windows)
- The home directory is located at `/home/johndoe`
- This remote experiment description uses the HSQLDB [4] database, which requires editing the *DatabaseUtils.props* file
- HSQLDB needs special parameter settings being transferred from *DatabaseUtils.props.hsql* to *DatabaseUtils.props*
- The jar archive for HSQLDB is located here: `/home/johndoe/hsqldb.jar`

Database Server Setup

To set up the database server, choose or create a directory to run the database server from, and start the server with:

```
java -classpath /home/johndoe/hsqldb.jar org.hsqldb.Server -database.0 -dbname xdb
```

Remote Engine Setup

- First, set up a directory for scripts and policy files:

  ```
  /home/johndoe/remote_engine
  ```

- Next, copy the `remoteEngine.jar` (from the Weka distribution; or build it from the sources with `ant remotejar`) to the `/home/johndoe/remote_engine` directory
- Create a script, called `/home/johndoe/remote_engine/startRemoteEngine`, with the following content (don't forget to make it executable with *chmod*, if you're using a flavour of Unix):

```
/path/to/your/jdk/bin/java -Xmx256m \
-classpath /home/johndoe/hsqldb.jar:remoteEngine.jar \
-Djava.security.policy=remote_engine.policy weka.experiment.RemoteEngine &
```

- Now we will start the remote engines (note that the same version of Java must be used for the Experimenter and remote engines) :
  - Edit the `remote_engine.policy` file in `/home/johndoe/remote_engine` by adding the lines

    ```
    permission java.io.FilePermission
    "/home/johndoe/-", "read";
    ```

    Note: In case the datasets are not located in the user's home then another file permission entry will be needed in the policy file.
  - For each machine you want to run an engine on:
    - `ssh` to the machine
    - *cd* to `/home/johndoe/remote_engine`
    - Run `/home/johndoe/startRemoteEngine` (to enable the remote engines to use more memory, modify the `-Xmx` option in the `startRemoteEngine` script)

Configuring the Experimenter

Now we will run the Experimenter:

- Create a directory for the experiment
- Copy the *DatabaseUtils.props.hsql* file to this directory and rename it to *DatabaseUtils.props* - a copy comes with your Weka distribution in `weka/experiment`
- Edit this file and change the "`jdbcURL=jdbc:hsqldb:hsql://servername`" entry to include the name of the machine that is running your database server (e.g., `jdbcURL=jdbc:hsqldb:hsql://dodo.company.com`)
- Now start the experimenter (inside this directory):

```
java -classpath /home/johndoe/hsqldb.jar \
     -Djava.rmi.server.codebase=file:/home/<path to your weka classes>/ \
     weka.gui.experiment.Experimenter
```

Now we will configure the experiment:

- First of all select the *Advanced* mode in the *Setup* tab
- Now choose the *DatabaseResultListener* in the *Destination* panel. Configure this result producer and supply the value **sa** for the username
- From the *Result generator* panel choose either the *CrossValidationResultProducer* or the *RandomSplitResultProducer* (these are the most commonly used ones) and then configure the remaining experiment details (e.g., datasets and classifiers)

- Now enable the *Distribute Experiment* panel by checking the tick box
- Click on the *Hosts* button and enter the names of the machines that you started remote engines on
- You can choose to distribute by run or dataset (try to get a balance)
- Save your experiment configuration
- Now start your experiment as you would do normally
- Check your results in the *Analyse* tab by clicking either the *Database* or *Experiment* buttons

Troubleshooting

- DON'T FORGET THE TRAILING SLASH - when you provide the Java startup option for the Experimenter:

  ```
  -Djava.rmi.server.codebase=file:/home/<path to your weka classes>/
  ```

  without it, you will get security access exceptions.

- If you get an error at the start of an experiment that looks a bit like this:

  ```
  01:13:19: RemoteExperiment (//blabla.company.com/RemoteEngine)
  (sub)experiment (datataset vineyard.arff) failed : java.sql.SQLException:
  Table already exists: EXPERIMENT_INDEX in statement [CREATE TABLE
  Experiment_index ( Experiment_type LONGVARCHAR, Experiment_setup
  LONGVARCHAR, Result_table INT )]
  ```

  ```
  01:13:19: dataset :vineyard.arff RemoteExperiment
  (//blabla.company.com/RemoteEngine) (sub)experiment (datataset
  vineyard.arff) failed : java.sql.SQLException: Table already
  exists: EXPERIMENT_INDEX in statement [CREATE TABLE Experiment_index
  ( Experiment_type LONGVARCHAR, Experiment_setup LONGVARCHAR, Result_table
  INT )]. Scheduling for execution on another host.
  ```

  then do not panic - this happens because multiple remote machines are trying to create the same table and are temporarily locked out - this will resolve itself so just leave your experiment running - in fact, it is a sign that the experiment is working!

- If you serialized an experiment and then modify your *DatabaseUtils.props* file due to an error (e.g., a missing type-mapping), the Experimenter will use the *DatabaseUtils.props* you had *at the time you serialized the experiment.* Keep in mind that the serialization process also serializes the *DatabaseUtils* class and therefore stored your props-file! This is another reason for storing your experiments as XML and not in a properietary binary format the Java serialization produces.

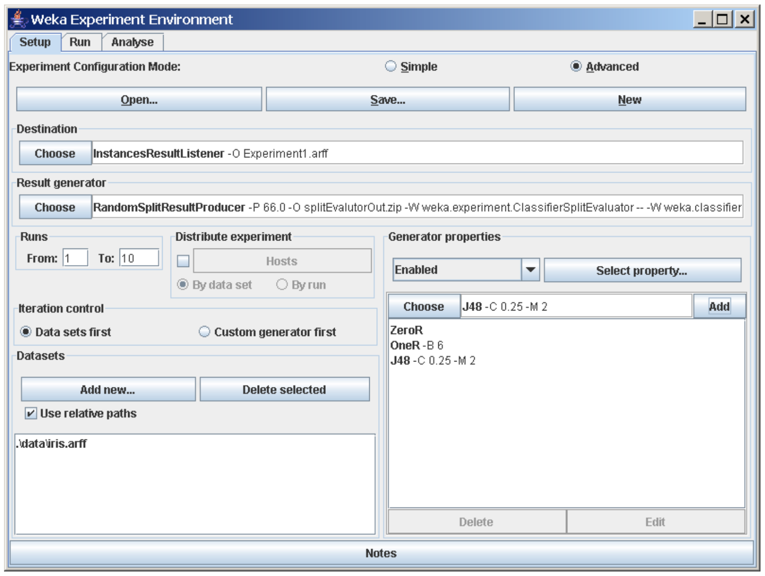
- Using a corrupt or incomplete *DatabaseUtils.props* file can cause peculiar interface errors, for example disabling the use of the "User" button alongside the database URL. If in doubt copy a clean *DatabaseUtils.props* from CVS.

- If you get `NullPointerException at java.util.Hashtable.get()` in the Remote Engine do not be alarmed. This will have no effect on the results of your experiment.

4 Analysing Results

Setup

Weka includes an experiment analyser that can be used to analyse the results of experiments that were sent to an *InstancesResultListener*. The experiment shown below uses 3 schemes, `ZeroR`, `OneR`, and `J48`, to classify the Iris data in an experiment using 10 train and test runs, with 66% of the data used for training and 34% used for testing.



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O Experiment1.arff

Result generator: RandomSplitResultProducer -P 66.0 -O splitEvalutorOut.zip -W weka.experiment.ClassifierSplitEvaluator -- -W weka.classifier

Runs From: 1 To: 10

Iteration control: [✓ Data sets first] Custom generator first

Datasets: [✓] Use relative paths

```
.\data\iris.arff
```

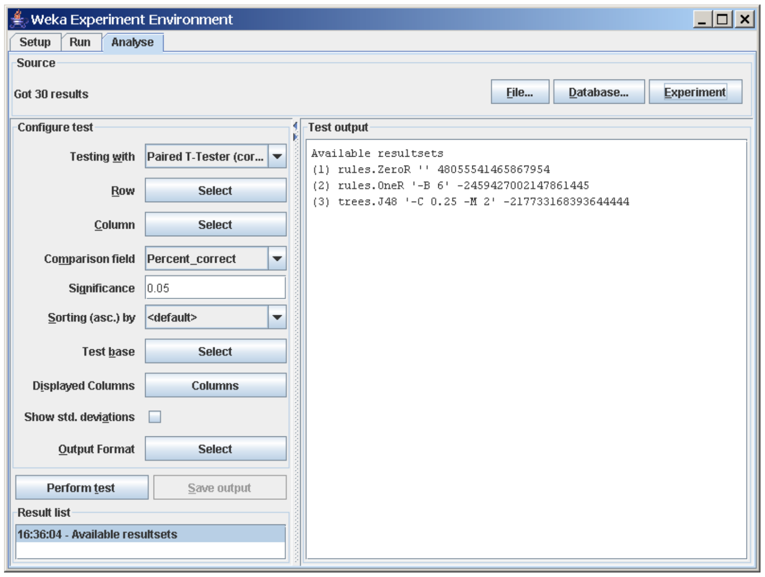
Generator properties: Enabled

J48 -C 0.25 -M 2

```
ZeroR
OneR -B 6
J48 -C 0.25 -M 2
```

After the experiment setup is complete, run the experiment. Then, to analyse the results, select the *Analyse* tab at the top of the Experiment Environment window.

Click on *Experiment* to analyse the results of the current experiment.



Got 30 results

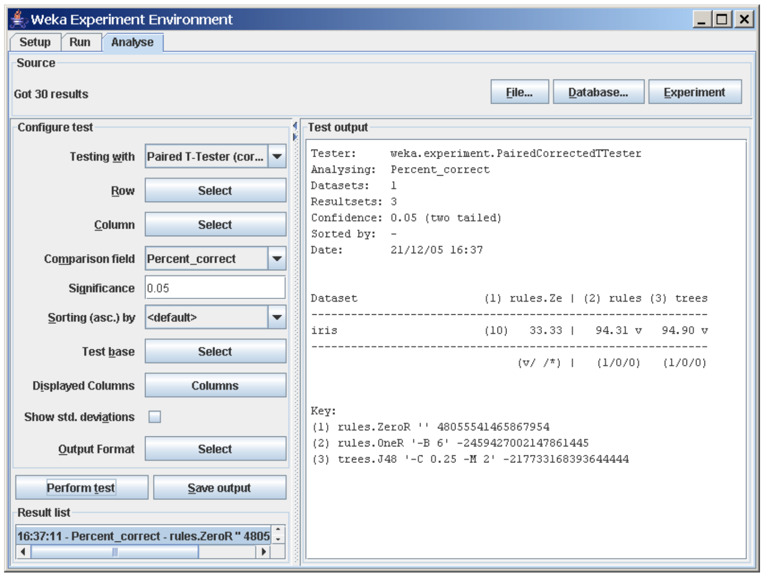
Testing with	Paired T-Tester (cor...
Row	Select
Column	Select
Comparison field	Percent_correct
Significance	0.05
Sorting (asc.) by	<default>
Test base	Select
Displayed Columns	Columns
Show std. deviations	☐
Output Format	Select

```
Available resultsets
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

16:36:04 - Available resultsets

The number of result lines available (*Got 30 results*) is shown in the *Source* panel. This experiment consisted of 10 runs, for 3 schemes, for 1 dataset, for a total of 30 result lines. Results can also be loaded from an earlier experiment file by clicking *File* and loading the appropriate *.arff* results file. Similarly, results sent to a database (using the *DatabaseResultListener*) can be loaded from the database.

Select the *Percent_correct* attribute from the *Comparison field* and click *Perform* test to generate a comparison of the 3 schemes.



Got 30 results

Testing with: Paired T-Tester (cor...
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ☐

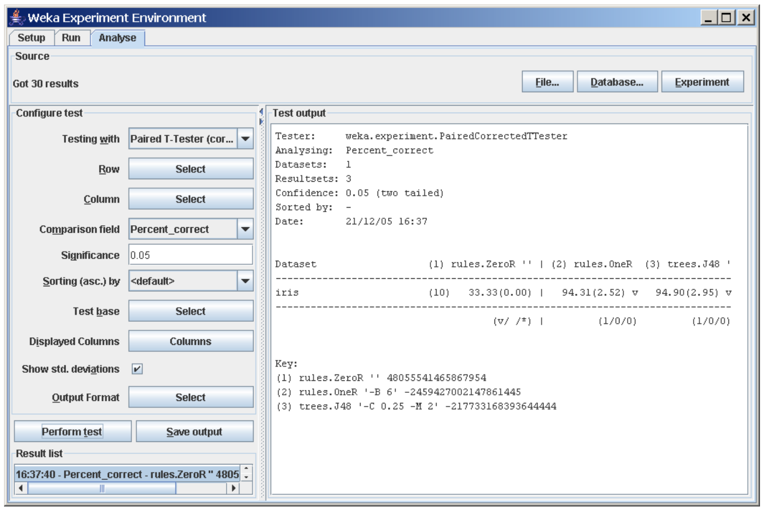
16:37:11 - Percent_correct - rules.ZeroR '' 4805

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:37


Dataset                   (1) rules.Ze | (2) rules (3) trees
------------------------------------------------------------
iris                      (10)   33.33 |   94.31 v   94.90 v
------------------------------------------------------------
                               (v/ /*) |   (1/0/0)   (1/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

The schemes used in the experiment are shown in the columns and the datasets used are shown in the rows.

The percentage correct for each of the 3 schemes is shown in each dataset row: 33.33% for `ZeroR`, 94.31% for `OneR`, and 94.90% for `J48`. The annotation v or * indicates that a specific result is statistically better (v) or worse (*) than the baseline scheme (in this case, `ZeroR`) at the significance level specified (currently 0.05). The results of both `OneR` and `J48` are statistically better than the baseline established by `ZeroR`. At the bottom of each column after the first column is a count (xx/ yy/ zz) of the number of times that the scheme was better than (xx), the same as (yy), or worse than (zz), the baseline scheme on the datasets used in the experiment. In this example, there was only one dataset and `OneR` was better than `ZeroR` once and never equivalent to or worse than `ZeroR` (1/0/0); `J48` was also better than `ZeroR` on the dataset.

The standard deviation of the attribute being evaluated can be generated by selecting the *Show std. deviations* check box and hitting *Perform test* again. The value *(10)* at the beginning of the *iris* row represents the number of estimates that are used to calculate the standard deviation (the number of runs in this case).



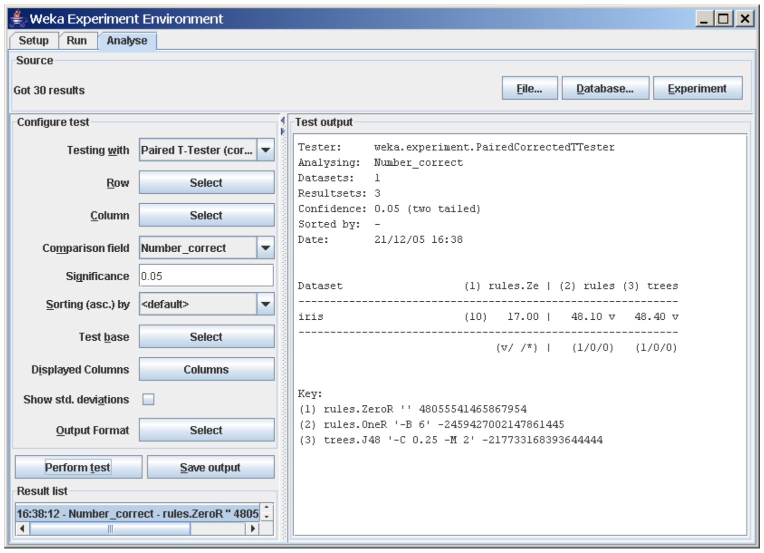
Got 30 results

Testing with: Paired T-Tester (cor...
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ✓

16:37:40 - Percent_correct - rules.ZeroR '' 4805

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:37


Dataset                   (1) rules.ZeroR '' | (2) rules.OneR  (3) trees.J48 '
-------------------------------------------------------------------------------
iris                      (10)   33.33(0.00) |   94.31(2.52) v   94.90(2.95) v
-------------------------------------------------------------------------------
                                     (v/ /*) |         (1/0/0)         (1/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

Selecting *Number_correct* as the comparison field and clicking *Perform test* generates the average number correct (out of 50 test patterns - 33% of 150 patterns in the Iris dataset).



Got 30 results

Testing with: Paired T-Tester (cor...
Comparison field: Number_correct
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ☐

16:38:12 - Number_correct - rules.ZeroR '' 4805

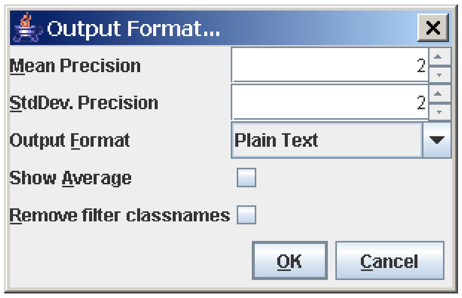
```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Number_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:38


Dataset                   (1) rules.Ze | (2) rules (3) trees
------------------------------------------------------------
iris                      (10)   17.00 |   48.10 v   48.40 v
------------------------------------------------------------
                               (v/ /*) |   (1/0/0)   (1/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

Clicking on the button for the *Output format* leads to a dialog that lets you choose the precision for the *mean* and the *std. deviations*, as well as the format of the output. Checking the *Show Average* checkbox adds an additional line to the output listing the average of each column. With the *Remove filter classnames* checkbox one can remove the filter name and options from processed datasets (filter names in Weka can be quite lengthy).

The following formats are supported:

- CSV
- GNUPlot
- HTML

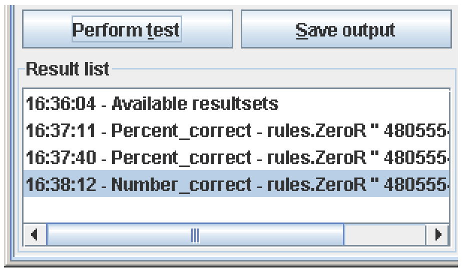
- LaTeX
- Plain text (default)
- Significance only



Mean Precision 2
StdDev. Precision 2
Output Format Plain Text
Show Average ☐
Remove filter classnames ☐

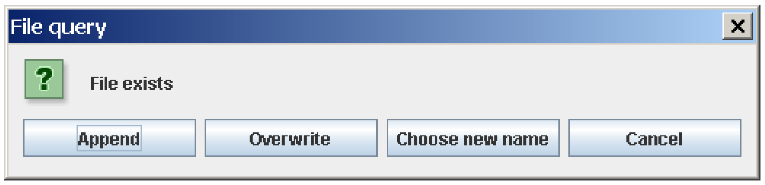
Saving the Results

The information displayed in the *Test output* panel is controlled by the currently-selected entry in the *Result list* panel. Clicking on an entry causes the results corresponding to that entry to be displayed.



```
16:36:04 - Available resultsets
16:37:11 - Percent_correct - rules.ZeroR '' 480555
16:37:40 - Percent_correct - rules.ZeroR '' 480555
16:38:12 - Number_correct - rules.ZeroR '' 480555
```

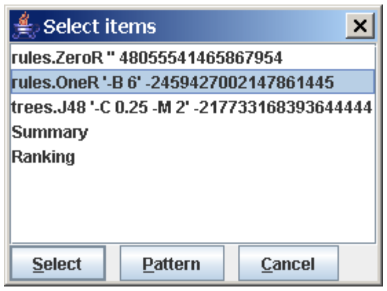
The results shown in the *Test output* panel can be saved to a file by clicking *Save output*. Only one set of results can be saved at a time but Weka permits the user to save all results to the same dataset by saving them one at a time and using the *Append* option instead of the *Overwrite* option for the second and subsequent saves.



File exists

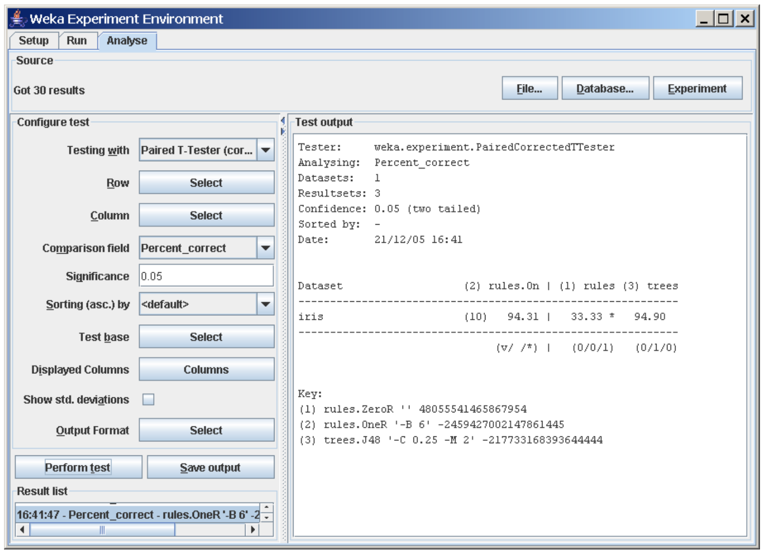
Changing the Baseline Scheme

The baseline scheme can be changed by clicking *Select base...* and then selecting the desired scheme. Selecting the `OneR` scheme causes the other schemes to be compared individually with the `OneR` scheme.



```
rules.ZeroR '' 48055541465867954
rules.OneR '-B 6' -2459427002147861445
trees.J48 '-C 0.25 -M 2' -217733168393644444
Summary
Ranking
```

If the test is performed on the *Percent_correct* field with `OneR` as the base scheme, the system indicates that there is no statistical difference between the results for `OneR` and `J48`. There is however a statistically significant difference between `OneR` and `ZeroR`.



Got 30 results

Testing with: Paired T-Tester (cor...
Row: Select
Column: Select
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Test base: Select
Displayed Columns: Columns
Show std. deviations
Output Format: Select

16:41:47 - Percent_correct - rules.OneR '-B 6' -2

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:41


Dataset                   (2) rules.On | (1) rules (3) trees
------------------------------------------------------------
iris                      (10)   94.31 |   33.33 *   94.90
------------------------------------------------------------
                               (v/ /*) |   (0/0/1)   (0/1/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) rules.OneR '-B 6' -2459427002147861445
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

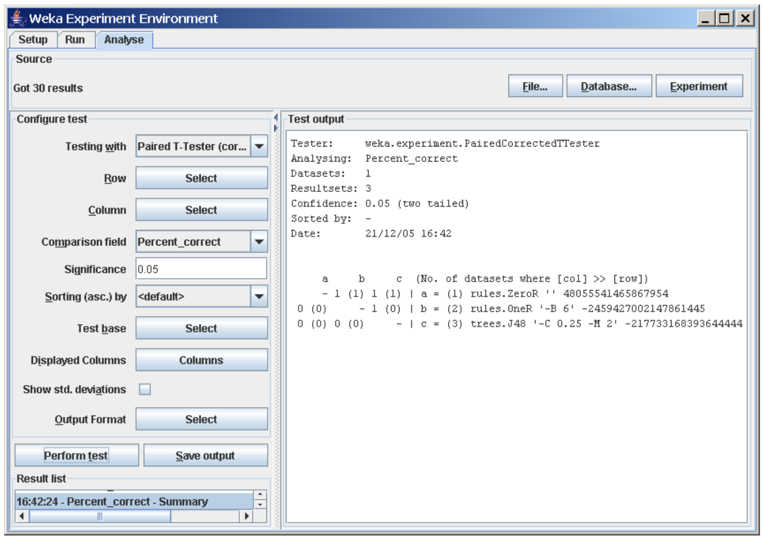
Statistical Significance

The term *statistical significance* used in the previous section refers to the result of a pair-wise comparison of schemes using either a standard *T-Test* or the corrected resampled *T-Test* [2]. The latter test is the default, because the standard *T-Test* can generate too many significant differences due to dependencies in the estimates (in particular when anything other than one run of an x-fold cross-validation is used). For more information on the *T-Test*, consult the Weka book [1] or an introductory statistics text. As the significance level is decreased, the confidence in the conclusion increases.

In the current experiment, there is not a statistically significant difference between the `OneR` and `J48` schemes.

Summary Test

Selecting *Summary* from *Test base* and performing a test causes the following information to be generated.



Got 30 results

Testing with: Paired T-Tester (cor...
Row: Select
Column: Select
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Test base: Select
Displayed Columns: Columns
Show std. deviations: ☐
Output Format: Select

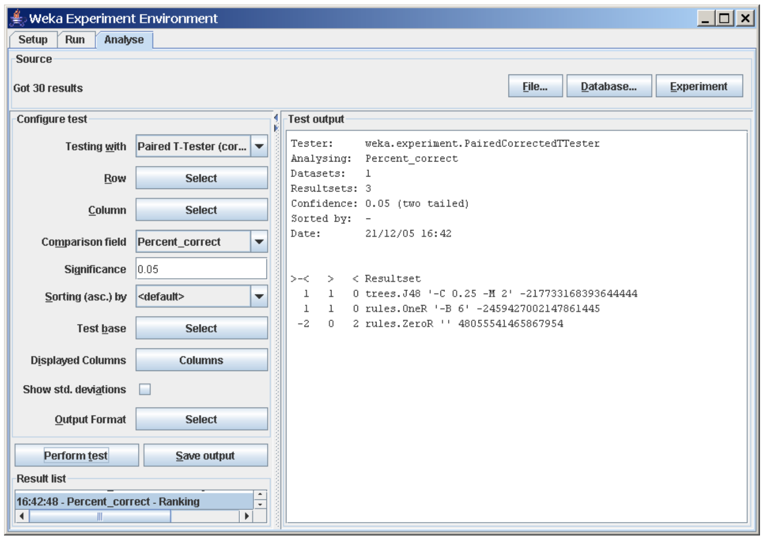
16:42:24 - Percent_correct - Summary

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:42


    a     b     c  (No. of datasets where [col] >> [row])
    - 1 (1) 1 (1) | a = (1) rules.ZeroR '' 48055541465867954
 0 (0)     - 1 (0) | b = (2) rules.OneR '-B 6' -2459427002147861445
 0 (0) 0 (0)     - | c = (3) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

In this experiment, the first row (- 1 1) indicates that column *b* (`OneR`) is better than row *a* (`ZeroR`) and that column *c* (`J48`) is also better than row *a*. The number in brackets represents the number of significant wins for the column with regard to the row. A 0 means that the scheme in the corresponding column did not score a single (significant) win with regard to the scheme in the row.

Ranking Test

Selecting *Ranking* from *Test base* causes the following information to be generated.



Got 30 results

Testing with: Paired T-Tester (cor...
Row: Select
Column: Select
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Test base: Select
Displayed Columns: Columns
Show std. deviations: ☐
Output Format: Select

16:42:48 - Percent_correct - Ranking

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       21/12/05 16:42


>-<   >   < Resultset
  1   1   0 trees.J48 '-C 0.25 -M 2' -217733168393644444
  1   1   0 rules.OneR '-B 6' -2459427002147861445
 -2   0   2 rules.ZeroR '' 48055541465867954
```

The ranking test ranks the schemes according to the total number of significant wins ($>$) and losses ($<$) against the other schemes. The first column ($> - <$) is the difference between the number of wins and the number of losses. This difference is used to generate the ranking.

References

[1] Witten, I.H. and Frank, E. (2005) *Data Mining: Practical machine learning tools and techniques. 2nd edition* Morgan Kaufmann, San Francisco.

[2] Bengio, Y. and Nadeau, C. (1999) *Inference for the Generalization Error.*

[3] Ross Quinlan (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, CA.

[4] *HSQLDB* – `http://hsqldb.sourceforge.net/`

[5] *WekaDoc* – `http://weka.sourceforge.net/wekadoc/`