Package 'Rdrools'

February 28, 2018

Type Package

Version 1.0.3

Date 2018-02-23

Title A Rules Engine Based on 'Drools'

Maintainer Ashwin Raaghav <Ashwin.Raaghav@mu-sigma.com>

Description An interface for using the popular Java based Drools, which is a Business Rule Management System (See <https://www.drools.org> for more information). This package allows you to run a set of rules written in DRL format on the data using the Drools engine. Credits to Mu Sigma for their continued support throughout the development of the package.

Depends R (>= 3.0.0), rJava, Rdroolsjars (>= 1.0.0)

SystemRequirements Java (>= 7.0)

License Apache License 2.0

LazyLoad yes

LazyData yes

NeedsCompilation no

Author Ashwin Raaghav [cre, aut],
SMS Chauhan [aut],
Zubin Dowlaty [aut],
Arushi Khattri [aut]

Repository CRAN

Date/Publication 2018-02-28 12:36:19 UTC

R topics documented:

Rdrools-package *A Rules Engine for R based on the popular Java based Drools*

Description

Rdrools is a robust rules engine based on the popular BRMS (Business Rule Management System) Drools written in Java. Rules make it easy to express solutions to different problems. They are often expressed as declarative elements of logic and are held in a file repository, separate from the production code, so that modifying the logic becomes easier. In Rdrools, rules are written in the native business readable DRL format, which is understood by drools. If you are not familiar, please go through the Drools documentation provided in the list of references below.

Details

Package:	Rdrools
Type:	Package
Version:	1.0.3
Date:	2018-02-23
License:	Apache License 2.0
LazyLoad:	yes
LazyData:	yes

Drools:	6.5.0.Final
JDK:	>= 1.7

Rdrools is fairly straightforward to use. It expects the user to provide a data frame as an input, a rules file, and a comma separated list of input and output columns. Rdrools then picks a row from the data frame, applies the rules provided to it and maps the output to the list of expected output columns. Incorrect inputs would throw a meaningful error. Rules syntax should be compatible with the drools version used in the package. The package does not require a separate drools installation.

Note

For applying a conditional rule, you can use the eval statement in your rules syntax. Please see a detailed example below.

```
import java.util.HashMap;
global java.util.HashMap output;
rule "xx"
    dialect "mvel"
    salience 3
    when
```

```
        input:HashMap()
        eval(input.get("currentAction")==input.get("previousAction"))
    then
        output.put("send","false")
end

rule "xy"
    dialect "mvel"
    salience 2
    when
        input:HashMap()
        eval(input.get("currentAction")!=input.get("previousAction"))
    then
        output.put("send","true")
end
```

Author(s)

Ashwin Raaghav <ashwin.raaghav@mu-sigma.com>, SMS Chauhan <sms.chauhan@mu-sigma.com>, Zubin Dowlaty <zubin.dowlaty@mu-sigma.com>, Arushi Khattri <arushi.khattri@mu-sigma.com>

References

Drools Homepage, Drools Documentation

See Also

rulesSession, runRules

Examples

```
library(Rdrools)
data(class)
data(rules)
input.columns<-c('name', 'class', 'grade', 'email')
output.columns<-c('address', 'subject', 'body')
rules.session<-rulesSession(rules, input.columns, output.columns)
output.df<-runRules(rules.session, class)
```

---

class *Sample dataset on which rules from the rules dataset can be applied*

Description

The dataset describes a class of students, their grades and contact details.

Usage

```
data(class)
```

Format

A data frame with 15 observations on the following 5 variables.

`id` a numeric vector

`name` a factor with levels `James` `Johnny` `Joseph`

`class` a factor with levels `English` `History` `Math` `Science` `Spanish`

`grade` a numeric vector

`email` a factor with levels `james@school.edu` `johhny@school.edu` `johnny@school.edu` `joseph@school.edu`

Examples

```
data(class)
str(class)
```

---

rules *A set of rules, for the* `class` *dataset*

Description

a character vector of lines read using `readLines()` from a drools rule file (text files with drools rules)

Usage

```
data(rules)
```

Format

The expected format is a character vector.

Details

You could create a similar rules character vector by using
`rules<-readLines('rules.drl')`

Examples

```
data(rules)
str(rules)
```

`rulesSession` *Creates a session of the rules engine*

Description

The rulesSession creates a session that interfaces between R and the Drools engine. The session is utilized by the runRules function for executing a data frame against a set of rules.

Usage

```
rulesSession(rules, input.columns, output.columns)
```

Arguments

`rules`	a character vector consisting of lines read from a rules file of format drl(drools rules file) This character vector is eventually collapsed into a character vector of length 1, so the way you read the file could potentially be just about anything
`input.columns`	a character vector of a set of input column, for example `input.columns<-c('name', 'class', 'grade', 'email')`
`output.columns`	a character vector of a set of expected output columns, for example `output.columns<-c('address', 'subject', 'body')`

Details

An active drools rules session. This promotes re-usability of a session, i.e. you can utilize the same session repetitively for different data sets of the same format.

Value

`rules.session.object`
Returns a session to the rules engine

Note

Please have a look at the examples provided in the 'examples' section of the `Rdrools` man page. A sample data set and a set of rules have been supplied help you understand the package usage.

Author(s)

Ashwin Raaghav <ashraaghav@gmail.com>, SMS Chauhan <smschauhah@gmail.com>

See Also

`runRules`, `Rdrools`

Examples

```
library(Rdrools)
data(rules)
input.columns<-c('name', 'class', 'grade', 'email')
output.columns<-c('address', 'subject', 'body')
rules.session<-rulesSession(rules, input.columns, output.columns)
```

`runRules` *Apply a set of rule transformations to a data frame*

Description

This function is the core of the Rdrools package. Rules are applied on an input data frame and the results are returned as the output of the function. The columns on which the rules need to be applied have to be provided explicitly. Additionally, the new columns that would be created based on the rules have to be provided explicitly as well.
The rules engine picks up a row from the dataframe, applies the transformation to it based on rules provided and saves the result in an output dataframe.

Usage

```
runRules(rules.session, input.df)
```

Arguments

`rules.session`	a session of the rules engine created using the the `rulesSession`
`input.df`	a dataframe consisting of a set of rows you wish to transform, and columns you wish to use in the transformation

Details

If you are not familiar with the drools file format, please have a look at the references provided in the `Rdrools` man page. More details on how conflicting rules are resolved using either salience or the Reete algorithm are also present in the references.

Value

`output.df`	a dataframe which is the result of transformations applied to the input dataframe(`input.df`), the columns being the list provided through the `output.columns` parameter in `rulesSession`.

Warning

Transformation policy
Transformations are applied row by row, iteratively. That is to say, all inputs required for a rule transformation should be present in columns as a part of that row itself. Each row should be considered independent of another; all input values required for a transformation should be available in that row itself. The expectation from rules engines are often misplaced.
Column Mismatch
Please make sure that the list of output columns provided through the `output.columns` parameter is exhaustive. Any additional column which is created through the rules transformation but is not present in the list would inhibit proper functioning. In most cases, an error should be thrown.

Author(s)

Ashwin Raaghav <ashraaghav@gmail.com>, SMS Chauhan <smschauhah@gmail.com>

See Also

Rdrools, rulesSession

Examples

```
library(Rdrools)
data(class)
data(rules)
input.columns<-c('name', 'class', 'grade', 'email')
output.columns<-c('address', 'subject', 'body')
rules.session<-rulesSession(rules, input.columns, output.columns)
output.df<-runRules(rules.session, class)
```

Index