# Package 'RKEEL'

September 14, 2023

Type Package

Title Using 'KEEL' in R Code
Version 1.3.4
<b>Depends</b> R (>= 3.4.0)
<b>Date</b> 2023-09-07
Author Jose M. Moyano [aut, cre], Luciano Sanchez [aut], Oliver Sanchez [ctb], Jesus Alcala-Fernandez [ctb]
Maintainer Jose M. Moyano <jmoyano1@us.es></jmoyano1@us.es>
Description  'KEEL' is a popular 'Java' software for a large number of different knowledge data discovery tasks. This package takes the advantages of 'KEEL' and R, allowing to use 'KEEL' algorithms in simple R code.  The implemented R code layer between R and 'KEEL' makes easy both using 'KEEL' algorithms in R as implementing new algorithms for 'RKEEL' in a very simple way. It includes more than 100 algorithms for classification, regression, preprocess, association rules and imbalance learning, which allows a more complete experimentation process. For more information about 'KEEL', see <a href="http://www.keel.es/">http://www.keel.es/</a> >.
SystemRequirements Java (>= 8)
License GPL
Imports R6, XML, doParallel, foreach, gdata, RKEELdata (>= 1.0.5), pmml, arules, Matrix, rJava, openssl, downloader
NeedsCompilation no
Repository CRAN
<b>Date/Publication</b> 2023-09-14 18:50:08 UTC
R topics documented:
ABB_IEP_FS

AdaBoost_I	
Alatasetal_A	8
Alcalaetal_A	12
AllKNN_TSS	15
AllPosible MV	16
ANR_F	17
Apriori_A	
ART C	
AssociationRulesAlgorithm	22
AssociativeClassificationAlgorithm	
Bayesian_D	
BNGE_C	
Bojarczuk_GP_C	
BSE_C	
C45Binarization_C	
C45Rules_C	
C45_C	
CamNN_C	
CART_C	
CART_R	
CBA_C	
CenterNN_C	
CFAR_C	
CFKNN_C	
CHC_C	
ClassificationAlgorithm	
ClassificationResults	
CleanAttributes_TR	
ClusterAnalysis_D	
CMAR C	
CNN C	
CPAR C	
CPW C	
CW C	
C_SVM_C	
DecimalScaling_TR	
DecrRBFN C	
Deeps_C	43
downloadFromMirror	
DSM_C	
DT_GA_C	
EARMGA_A	48
Eclat_A	
EPSILON_SVR_R	
Falco_GP_C	
FCRA_C	
FPgrowth_A	
FRNN_C	61

FRSBM_R	62
FURIA_C	63
FuzzyApriori_A	64
FuzzyFARCHD_C	67
FuzzyKNN_C	68
FuzzyNPC_C	69
GANN_C	70
GAR_A	72
GENAR_A	76
GeneticFuzzyAprioriDC_A	79
GeneticFuzzyApriori_A	83
getAttributeLinesFromDataframes	87
getExePath	88
getJarList	88
getJarPath	89
GFS_AdaBoost_C	89
GFS_GP_R	90
GFS_GSP_R	91
	91
GFS_LogitBoost_C	93
GFS_RB_MF_R	
hasContinuousData	95
hasMissing Values	95
ID3_C	96
ID3_D	96
IF_KNN_C	97
Ignore_MV	98
ImbalancedClassificationAlgorithm	
IncrRBFN_C	
isMultiClass	
IterativePartitioningFilter_F	100
JFKNN_C	101
KeelAlgorithm	102
Kernel_C	102
KMeans_MV	103
KNN_C	104
KNN_MV	104
KSNN C	
KStar C	106
LDA_C	
LinearLMS C	
LinearLMS R	
loadKeelDataset	
Logistic_C	
LVF_IEP_FS	
M5Rules R	
M5 R	
MinMax_TR	
MLP_BP_C	114

MLP_BP_R	
ModelCS_TSS	116
MODENAR_A	117
MOEA_Ghosh_A	121
MOPNAR_A	124
MostCommon_MV	
NB_C	
NICGAR_A	
NM_C	
NNEP C	134
Nominal2Binary_TR	135
NU_SVM_C	136
NU_SVR_R	
PART_C	
PDFC_C	
PFKNN_C	
PNN C	
PolQuadraticLMS_C	
PolQuadraticLMS_R	
POP_TSS	
PreprocessAlgorithm	
PRISM_C	
Proportional_D	
PSO_ACO_C	
PSRCG_TSS	
PUBLIC_C	
PW C	
QAR_CIP_NSGAII_A	
QDA C	
RBFN C	
RBFN R	
read.keel	156
RegressionAlgorithm	
RegressionResults	
Relief FS	
Ripper_C	158
RISE C	
runCV	159
runParallel	160
runSequential	161
SaturationFilter F	
SFS_IEP_FS	163
SGA C	
Shrink_C	
Slipper_C	
SMO C	
SSGA_Integer_knn_FS	
Tan_GP_C	

ABB\_IEP\_FS 5

	Thrift_R																		 			170
	UniformFrequency_D .																		 			171
	UniformWidth_D																		 			172
	VWFuzzyKNN_C																		 			173
	WM_R																		 			173
	writeDatFromDataframe																		 			174
	writeDatFromDataframes	·																	 			175
	ZScore_TR																		 		•	175
Index																						177
ABB_I	IEP_FS ABI		P_	FS	KE	EL	Pi	rep	ro	ces	SS 2	Al <sub>ξ</sub>	 goi	riti	hm							

# Description

ABB\_IEP\_FS Preprocess Algorithm from KEEL.

# Usage

```
ABB_IEP_FS(train, test, seed)
```

# **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

# Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ABB_IEP_FS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

6 AdaBoostNC\_C

# **Description**

AdaBoostNC\_C Classification Algorithm from KEEL.

# Usage

```
AdaBoostNC_C(train, test, pruned, confidence, instancesPerLeaf, numClassifiers, algorithm, trainMethod, lambda, seed)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pruned pruned. Default value = TRUE
confidence confidence. Default value = 0.25

instancesPerLeaf

instancesPerLeaf. Default value = 2 numClassifiers numClassifiers. Default value = 10

algorithm algorithm. Default value = "ADABOOST.NC" trainMethod trainMethod. Default value = "NORESAMPLING"

lambda. Default value = 2

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::AdaBoostNC_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

AdaBoost\_I 7

AdaBoost I	AdaBoost_I KEEL Imbalanced	Classification Algorithm
/ (ddb000 t_1	THE BE INTO CHANGE OF	

# **Description**

AdaBoost\_I Imbalanced Classification Algorithm from KEEL.

## Usage

```
AdaBoost_I(train, test, pruned, confidence, instancesPerLeaf, numClassifiers, algorithm, trainMethod, seed)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pruned pruned. Default value = TRUE
confidence confidence. Default value = 0.25

instancesPerLeaf

instancesPerLeaf. Default value = 2 numClassifiers numClassifiers. Default value = 10

algorithm algorithm. Default value = "ADABOOST"

trainMethod trainMethod. Default value = "NORESAMPLING"

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::AdaBoost_I(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Alatasetal\_A

Alatasetal\_A KEEL Association Rules Algorithm

# **Description**

Alatasetal\_A Association Rules Algorithm from KEEL.

# Usage

Alatasetal\_A(dat, seed, NumberofEvaluations, InitialRandomChromosomes, rDividingPoints, TournamentSize, ProbabilityofCrossover, MinimumProbabilityofMutation, MaximumProbabilityofMutation, ImportanceofRulesSupport, ImportanceofRulesConfidence, ImportanceofNumberofInvolvedAttributes, ImportanceofIntervalsAmplitude, ImportanceofNumberofRecordsAlreadyCovered, AmplitudeFactor)

# **Arguments**

dat Dataset as a data.frame object

seed seed. Default value = 1286082570

NumberofEvaluations

Number of Evaluations. Default value = 50000

InitialRandomChromosomes

Initial Random Chromosomes. Default value = 12

rDividingPoints

r-Dividing Points. Default value = 3

TournamentSize TournamentSize. Default value = 10

ProbabilityofCrossover

Probability of Crossover. Default value = 0.7

MinimumProbabilityofMutation

Minimum Probability of Mutation. Default value = 0.05

MaximumProbabilityofMutation

Maximum Probability of Mutation. Default value = 0.9

 ${\tt Importance of Rules Support}$ 

Importance of Rules Support. Default value = 5

ImportanceofRulesConfidence

Importance of Rules Confidence. Default value = 20

ImportanceofNumberofInvolvedAttributes

Importance of Number of Involved Attributes. Default value = 0.05

 $Importance of Intervals {\tt Amplitude}$ 

Importance of Intervals Amplitude. Default value = 0.02

 $Importance of {\tt Number of Records Already Covered}$ 

Importance of Number of Records Already Covered. Default value = 0.01

AmplitudeFactor

Amplitude Factor. Default value = 2.0

#### **Details**

```
$run() Run algorith
$showRules(numRules) Show a number of rules. By default all rules.
$getInterestMeasures() Return a data.frame with all interest measures of set rules.
$sortBy(interestMeasure) Order set rules by interest measure.
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
```

```
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
```

```
"sebag", Sebag measure (Sebag and Schoenauer, 1988)

"support", supp (Agrawal et al., 1996)

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)

"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)
```

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::Alatasetal_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

12 Alcalaetal\_A

Alcalaetal\_A

Alcalaetal\_A KEEL Association Rules Algorithm

# **Description**

Alcalaetal\_A Association Rules Algorithm from KEEL.

#### Usage

Alcalaetal\_A(dat, seed, NumberofEvaluations, PopulationSize, NumberofBitsperGene, DecreasingFactorofLthresholdNOTUSED, FactorforParentCentricBLXCrossover, NumberofFuzzyRegionsforNumericAttributes, UseMaxOperatorfor1FrequentItemsets, MinimumSupport, MinimumConfidence)

#### Arguments

dat Dataset as a data.frame object

seed seed. Default value = 1286082570

NumberofEvaluations

Number of Evaluations. Default value = 10000

PopulationSize Population Size. Default value = 50

NumberofBitsperGene

Number of Bits per Gene. Default value = 30

 ${\tt DecreasingFactorofLthresholdNOTUSED}$ 

Decreasing Factor of Lthreshold NOT USED. Default value = 0.1

FactorforParentCentricBLXCrossover

Factor for Parent Centric BLXCrossover. Default value = 1.0

 ${\tt Number of Fuzzy Regions for Numeric Attributes}$ 

Number of Fuzzy Regions for Numeric Attributes. Default value = 3

 ${\tt Use Max Operator for 1 Frequent Items ets}$ 

Use Max Operator for 1 Frequent Itemsets. Default value = "false"

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

```
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
```

14 Alcalaetal\_A

```
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
```

AllKNN\_TSS 15

```
"yuleY", Yule's Y (Tan and Kumar, 2000)
```

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

#### **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::Alcalaetal_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

AllKNN\_TSS

AllKNN\_TSS KEEL Preprocess Algorithm

#### **Description**

AllKNN\_TSS Preprocess Algorithm from KEEL.

# Usage

```
AllKNN_TSS(train, test, k, distance)
```

16 AllPosible\_MV

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k. Default value = 3

distance distance. Default value = "Euclidean"

#### Value

A data.frame with the preprocessed data for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::AllKNN_TSS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

AllPosible\_MV

AllPosible\_MV KEEL Preprocess Algorithm

## **Description**

AllPosible\_MV Preprocess Algorithm from KEEL.

#### Usage

```
AllPosible_MV(train, test)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

ANR\_F 17

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::AllPosible_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

ANR\_F

ANR\_F KEEL Preprocess Algorithm

## **Description**

ANR\_F Preprocess Algorithm from KEEL.

# Usage

```
ANR_F(train, test, seed)
```

# **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

# Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("zoo")
data_test <- RKEEL::loadKeelDataset("zoo")

#Create algorithm
algorithm <- RKEEL::ANR_F(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

18 Apriori\_A

Apriori\_A

Apriori\_A KEEL Association Rules Algorithm

# **Description**

Apriori\_A Association Rules Algorithm from KEEL.

#### Usage

Apriori\_A(dat, NumberofPartitionsforNumericAttributes, MinimumSupport,
 MinimumConfidence)

# **Arguments**

dat Dataset as a data.frame object

 ${\tt Number of Partitions for Numeric Attributes}$ 

Number of Partitions for Numeric Attributes. Default value = 4

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

#### **Details**

```
$run() Run algorith
```

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

```
"allConfidence" (Omiencinski, 2003)
```

<sup>&</sup>quot;crossSupportRatio", cross-support ratio (Xiong et al., 2003)

<sup>&</sup>quot;lift", interest factor (Brin et al. 1997)

<sup>&</sup>quot;support", supp (Agrawal et al., 1996)

Apriori\_A

```
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
```

20 Apriori\_A

```
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")
#Create algorithm</pre>
```

*ART\_C* 21

```
algorithm <- RKEEL::Apriori_A(dat)

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

#Show a number of rules
algorithm$showRules(2)

#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()

#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY","yulesY")

#Sort by interest measure lift
algorithm$sortBy("lift")

#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))</pre>
```

ART\_C

ART\_C KEEL Classification Algorithm

# **Description**

ART\_C Classification Algorithm from KEEL.

# Usage

```
ART_C(train, test)
```

# Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

# Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")
#Create algorithm</pre>
```

```
algorithm <- RKEEL::ART_C(data_train, data_test)
#Run algorithm
algorithm$run()
#See results
algorithm$testPredictions</pre>
```

AssociationRulesAlgorithm

Association Rules Algorithm

#### **Description**

Class inheriting of KeelAlgorithm, to common methods for all KEEL Association Rules Algorithms. The specific association rules algorithms must inherit of this class.

The run() method receives three parameters. The folderPath parameter indicates where to place the folder with the experiments if wanted. If it is not indicated, the folder is placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

AssociativeClassificationAlgorithm

Associative Classification Algorithm

#### Description

Class inheriting of ClassificationAlgorithm, to common methods for Associative Classification Algorithms.

The run() method receives three parameters. The folderPath parameter indicates where to place the folder with the experiments if wanted. If it is not indicated, the folder is placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

Bayesian\_D 23

Bayesian\_D

Bayesian\_D KEEL Preprocess Algorithm

# Description

Bayesian\_D Preprocess Algorithm from KEEL.

# Usage

```
Bayesian_D(train, test)
```

# **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

## Value

A data.frame with the preprocessed data for both train and test datasets.

# **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::Bayesian_D(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

BNGE\_C

BNGE\_C KEEL Classification Algorithm

# Description

BNGE\_C Classification Algorithm from KEEL.

# Usage

```
BNGE_C(train, test, seed)
```

24 Bojarczuk\_GP\_C

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::BNGE_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Bojarczuk\_GP\_C

Bojarczuk\_GP\_C KEEL Classification Algorithm

#### **Description**

Bojarczuk\_GP\_C Classification Algorithm from KEEL.

#### Usage

```
Bojarczuk_GP_C(train, test, population_size, max_generations,
    max_deriv_size, rec_prob, copy_prob, seed)
```

#### **Arguments**

train Train dataset as a data.frame object

test Test dataset as a data.frame object

population\_size

population\_size. Default value = 200

max\_generations

max\_generations. Default value = 200

max\_deriv\_size max\_deriv\_size. Default value = 20

rec\_prob rec\_prob. Default value = 0.8

BSE\_C 25

copy\_prob copy\_prob. Default value = 0.01

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Bojarczuk_GP_C(data_train, data_test)
algorithm <- RKEEL::Bojarczuk_GP_C(data_train, data_test, population_size=5, max_generations=10)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

BSE\_C

BSE\_C KEEL Classification Algorithm

## **Description**

BSE\_C Classification Algorithm from KEEL.

## Usage

```
BSE_C(train, test, k, distance)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k. Default value = 1

distance distance. Default value = "Euclidean"

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

26 C45Binarization\_C

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::BSE_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

C45Binarization\_C

C45Binarization\_C KEEL Classification Algorithm

## **Description**

C45Binarization\_C Classification Algorithm from KEEL.

#### Usage

```
C45Binarization_C(train, test, pruned, confidence, instancesPerLeaf, binarization, scoreFunction, bts)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pruned pruned. Default value = TRUE
confidence confidence. Default value = 0.25
instancesPerLeaf
instancesPerLeaf. Default value = 2
binarization binarization. Default value = "OVO"

 ${\tt scoreFunction} \quad {\tt scoreFunction}. \ Default \ value = "WEIGHTED"$ 

bts. Default value = 0.05

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

C45Rules\_C

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::C45Binarization_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

C45Rules\_C

C45Rules\_C KEEL Classification Algorithm

#### **Description**

C45Rules\_C Classification Algorithm from KEEL.

#### Usage

```
C45Rules_C(train, test, confidence, itemsetsPerLeaf, threshold, seed)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
confidence confidence. Default value = 0.25
itemsetsPerLeaf

itemsetsPerLeaf. Default value = 2

threshold threshold. Default value = 10

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

 $A \ data. frame \ with \ the \ actual \ and \ predicted \ classes \ for \ both \ train \ and \ test \ datasets.$ 

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::C45Rules_C(data_train, data_test)</pre>
```

28 C45\_C

```
#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

C45\_C

C45\_C KEEL Classification Algorithm

# Description

C45\_C Classification Algorithm from KEEL.

## Usage

```
C45_C(train, test, pruned, confidence, instancesPerLeaf)
```

#### Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pruned pruned. Default value = TRUE
confidence confidence. Default value = 0.25
instancesPerLeaf
instancesPerLeaf. Default value = 2

mounteest etzeut. Detaut vara

# Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::C45_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CamNN\_C

 ${\tt CamNN\_C}$ 

CamNN\_C KEEL Classification Algorithm

#### **Description**

CamNN\_C Classification Algorithm from KEEL.

# Usage

```
CamNN_C(train, test, k)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
k Default value = 1

# Value

A data.frame with the actual and predicted classes for both train and test datasets.

# **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CamNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CART\_C

CART\_C KEEL Classification Algorithm

# **Description**

CART\_C Classification Algorithm from KEEL.

#### Usage

```
CART_C(train, test, maxDepth)
```

30 CART\_R

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

maxDepth k. Default value = 90

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CART_C(data_train, data_test, maxDepth=3)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CART\_R

CART\_R KEEL Regression Algorithm

# Description

CART\_R Regression Algorithm from KEEL.

## Usage

```
CART_R(train, test, maxDepth)
```

# **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
maxDepth maxDepth. Default value = 90

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

CBA\_C 31

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::CART_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CBA\_C

CBA\_C KEEL Associative Classification Algorithm

#### **Description**

CBA\_C Associative Classification Algorithm from KEEL.

#### Usage

```
CBA_C(train, test, min_support, min_confidence, pruning, maxCandidates)
```

#### Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object min\_support min\_support. Default value = 0.01 min\_confidence min\_confidence. Default value = 0.5

pruning indicates wether pruning or not. Default value = TRUE maxCandidates maxCandidates; if 0, no limit. Default value = 80000

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data <- loadKeelDataset("breast")

#Create algorithm
algorithm <- RKEEL::CBA_C(data, data)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CFAR\_C

CenterNN\_C

CenterNN\_C KEEL Classification Algorithm

# Description

CenterNN\_C Classification Algorithm from KEEL.

# Usage

```
CenterNN_C(train, test)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CenterNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CFAR\_C

CFAR\_C KEEL Classification Algorithm

# Description

CFAR\_C Classification Algorithm from KEEL.

# Usage

```
CFAR_C(train, test, min_support, min_confidence, threshold,
   num_labels, seed)
```

CFKNN\_C 33

# **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
min\_support min\_support. Default value = 0.1
min\_confidence min\_confidence. Default value = 0.85
threshold threshold. Default value = 0.15
num\_labels num\_labels. Default value = 5

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CFAR_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CFKNN\_C

CFKNN\_C KEEL Classification Algorithm

#### **Description**

CFKNN\_C Classification Algorithm from KEEL.

# Usage

```
CFKNN_C(train, test, k, alpha, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k k. Default value = 3 alpha alpha. Default value = 0.6

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

34 CHC\_C

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CFKNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CHC\_C

CHC\_C KEEL Classification Algorithm

#### **Description**

CHC\_C Classification Algorithm from KEEL.

# Usage

```
CHC_C(train, test, pop_size, evaluations, alfa, restart_change,
    prob_restart, prob_diverge, k, distance, seed)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pop\_size pop\_size. Default value = 50
evaluations evaluations. Default value = 10000

alfa alfa. Default value = 0.5

restart\_change restart\_change. Default value = 0.35
prob\_restart prob\_restart. Default value = 0.25
prob\_diverge prob\_diverge. Default value = 0.05

k k. Default value = 1

distance distance. Default value = "Euclidean"

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CHC_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

ClassificationAlgorithm

Classification Algorithm

#### **Description**

Class inheriting of KeelAlgorithm, to common methods for all KEEL Classification Algorithms. The specific classification algorithms must inherit of this class.

The run() method receives three parameters. The folderPath parameter indicates where to place the folder with the experiments if wanted. If it is not indicated, the folder is placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

ClassificationResults Classification Results

## **Description**

Class to calculate and store some results for a ClassificationAlgorithm. It receives as parameter the prediction of a classification algorithm as a data.frame object.

36 ClusterAnalysis\_D

 ${\tt CleanAttributes\_TR} \quad {\tt CleanAttributes\_TR} \; {\tt KEEL\ Preprocess\ Algorithm}$ 

# Description

CleanAttributes\_TR Preprocess Algorithm from KEEL.

#### Usage

```
CleanAttributes_TR(train, test)
```

# **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::CleanAttributes_TR(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

ClusterAnalysis\_D

ClusterAnalysis\_D KEEL Preprocess Algorithm

# Description

ClusterAnalysis\_D Preprocess Algorithm from KEEL.

# Usage

```
ClusterAnalysis_D(train, test)
```

CMAR\_C 37

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ClusterAnalysis_D(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

CMAR\_C

CMAR\_C KEEL Associative Classification Algorithm

#### **Description**

CMAR\_C Associative Classification Algorithm from KEEL.

## Usage

```
CMAR_C(train, test, min_confidence, min_support, databaseCoverage)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
min\_confidence min\_confidence. Default value = 0.5
min\_support min\_support. Default value = 0.01
databaseCoverage
databaseCoverage. Default value = 4

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

38 *CNN\_C* 

#### **Examples**

```
data <- loadKeelDataset("breast")

#Create algorithm
algorithm <- RKEEL::CMAR_C(data, data)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

CNN\_C

CNN\_C KEEL Classification Algorithm

# Description

CNN\_C Classification Algorithm from KEEL.

### Usage

```
CNN_C(train, test, k, distance, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k. Default value = 1

distance distance. Default value = "Euclidean"

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $CPAR\_C$  39

CPAR\_C

CPAR\_C KEEL Associative Classification Algorithm

# Description

CPAR\_C Associative Classification Algorithm from KEEL.

# Usage

```
CPAR_C(train, test, delta, min_gain, alpha, rules_prediction)
```

# **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
delta	delta. Default value = 0.05
min_gain	min_gain. Default value = 0.7
alpha	alpha. Default value = 0.66
rules_prediction	
	rules_prediction. Default value = 5

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data <- loadKeelDataset("breast")

#Create algorithm
algorithm <- RKEEL::CPAR_C(data, data)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

40 *CPW\_C* 

CPW\_C

# CPW\_C KEEL Classification Algorithm

## **Description**

CPW\_C Classification Algorithm from KEEL.

## Usage

```
CPW_C(train, test, beta, mu, ro, epsilon)
```

## **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
beta	beta. Default value = 8.0
mu	mu. Default value = 0.001
ro	ro. Default value = 0.001
epsilon	epsilon. Default value = 0.001

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CPW_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

*CW\_C* 41

 $CW\_C$ 

CW\_C KEEL Classification Algorithm

# Description

CW\_C Classification Algorithm from KEEL.

# Usage

```
CW_C(train, test, beta, mu, epsilon)
```

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
beta beta. Default value = 8.0
mu mu. Default value = 0.001
epsilon epsilon. Default value = 0.001

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::CW_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $C_SVM_C$ 

C\_SVM\_C

C\_SVM\_C KEEL Classification Algorithm

#### **Description**

C\_SVM\_C Classification Algorithm from KEEL.

## Usage

```
C_SVM_C(train, test, KernelType, C, eps, degree, gamma, coef0,
   nu, p, shrinking, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
KernelType KernelType. Default value = "RBF"

C C. Default value = 100.0

eps eps. Default value = 0.001

degree degree. Default value = 1

gamma gamma. Default value = 0.01

coef0 coef0. Default value = 0.0

nu nu. Default value = 0.1

p Default value = 1.0

shrinking shrinking. Default value = 1

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::C_SVM_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

DecimalScaling\_TR 43

# Description

DecimalScaling\_TR Preprocess Algorithm from KEEL.

#### Usage

```
DecimalScaling_TR(train, test)
```

## Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::DecimalScaling_TR(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

DecrRBFN\_C

DecrRBFN\_C KEEL Classification Algorithm

## **Description**

DecrRBFN\_C Classification Algorithm from KEEL.

## Usage

```
DecrRBFN_C(train, test, percent, num_neurons_ini, alfa, seed)
```

Deeps\_C

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

percent. Default value = 0.1

num\_neurons\_ini

num\_neurons\_ini. Default value = 20

alfa alfa. Default value = 0.3

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::DecrRBFN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Deeps\_C

Deeps\_C KEEL Classification Algorithm

## **Description**

Deeps\_C Classification Algorithm from KEEL.

# Usage

```
Deeps_C(train, test, beta)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

beta beta. Default value = 0.12

downloadFromMirror 45

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Deeps_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $\\ download From \\ Mirror$ 

Download file from a mirror

## **Description**

Downloads a file from a given mirror and checks its md5 sum. The file is stored in a given path

## Usage

```
downloadFromMirror(mirror, file_path, md5_sum)
```

#### Arguments

mirror URL from which to download the file.

file\_path Path or folder where the downloaded file will be stored.

md5\_sum md5 checksum string corresponding to the file to download. The method will

check that the downloaded file checksum and the md5\_sum parameter match.

#### Value

Returns 1 if the download was successful and -1 otherwise.

```
# Download RKEELjars file
dCode = RKEEL::downloadFromMirror("https://personal.us.es/jmoyano1/RKEELjars_1.1.zip",
    downloadedJarFile, md5_sum)
# Check if the download was successful
if(dCode<0){</pre>
```

DSM\_C

```
print('There was an error during the download.')
}
```

DSM\_C

DSM\_C KEEL Classification Algorithm

# Description

DSM\_C Classification Algorithm from KEEL.

## Usage

```
DSM_C(train, test, iterations, percentage, alpha_0, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object iterations iterations. Default value = 100 percentage percentage. Default value = 10 alpha\_0 alpha\_0. Default value = 0.1

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::DSM_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $DT\_GA\_C$  47

DT\_GA\_C

DT\_GA\_C KEEL Classification Algorithm

#### **Description**

DT\_GA\_C Classification Algorithm from KEEL.

### Usage

```
DT_GA_C(train, test, confidence, instancesPerLeaf,
    geneticAlgorithmApproach, threshold, numGenerations,
    popSize, crossoverProb, mutProb, seed)
```

#### Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object confidence confidence. Default value = 0.25

instancesPerLeaf

instancesPerLeaf. Default value = 2

geneticAlgorithmApproach

geneticAlgorithmApproach. Default value = "GA-LARGE-SN"

threshold threshold. Default value = 10

numGenerations numGenerations. Default value = 50

popSize popSize. Default value = 200 crossoverProb crossoverProb. Default value = 0.8 mutProb mutProb. Default value = 0.01

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::DT_GA_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

48 EARMGA A

EARMGA\_A

## EARMGA\_A KEEL Association Rules Algorithm

#### **Description**

EARMGA\_A Association Rules Algorithm from KEEL.

#### Usage

EARMGA\_A(dat, seed, FixedLengthofAssociationRules, PopulationSize,
 TotalNumberofEvaluations, DifferenceBoundaryNOTUSED, ProbabilityofSelection,
 ProbabilityofCrossover, ProbabilityofMutation,
 NumberofPartitionsforNumericAttributes)

### **Arguments**

dat Dataset as a data.frame object

seed seed. Default value = 1286082570

 ${\tt FixedLengthofAssociationRules}$ 

Fixed Length of Association Rules. Default value = 2

PopulationSize PopulationSize. Default value = 100

TotalNumberofEvaluations

Total Number of Evaluations. Default value = 50000

DifferenceBoundaryNOTUSED

Difference Boundary NOT USED. Default value = 0.01

ProbabilityofSelection

Probability of Selection. Default value = 0.75

ProbabilityofCrossover

Probability of Crossover. Default value = 0.7

ProbabilityofMutation

Probability of Mutation. Default value = 0.1

NumberofPartitionsforNumericAttributes

Number of Partitions for Numeric Attributes. Default value = 4

### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

EARMGA\_A 49

```
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
```

 $EARMGA\_A$ 

```
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
```

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

## **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::EARMGA_A(dat)
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

Eclat\_A

Eclat\_A KEEL Association Rules Algorithm

## Description

Eclat\_A Association Rules Algorithm from KEEL.

#### Usage

```
Eclat_A(dat, NumberofPartitionsforNumericAttributes, MinimumSupport,
    MinimumConfidence)
```

## Arguments

#### **Details**

```
$run() Run algorith
$showRules(numRules) Show a number of rules. By default all rules.
$getInterestMeasures() Return a data.frame with all interest measures of set rules.
$sortBy(interestMeasure) Order set rules by interest measure.
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
```

```
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
```

```
"mutualInformation", uncertainty, M (Tan et al., 2002)

"oddsRatio", odds ratio alpha (Tan et al., 2004)

"phi", correlation coefficient phi (Tan et al. 2004)

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)

"sebag", Sebag measure (Sebag and Schoenauer, 1988)

"support", supp (Agrawal et al., 1996)

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)

"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)

For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")

#Create algorithm
algorithm <- RKEEL::Eclat_A(dat)

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

#Show a number of rules
algorithm$showRules(2)

#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()

#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")</pre>
```

EPSILON\_SVR\_R 55

```
#Sort by interest measure lift
algorithm$sortBy("lift")

#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

EPSILON\_SVR\_R

EPSILON\_SVR\_R KEEL Regression Algorithm

# Description

EPSILON\_SVR\_R Regression Algorithm from KEEL.

#### Usage

```
EPSILON_SVR_R(train, test, KernelType, C, eps, degree, gamma,
  coef0, nu, p, shrinking, seed)
```

Train dataset as a data.frame object

#### **Arguments**

train

Test dataset as a data.frame object test KernelType KernelType. Default value = "RBF" C. Default value = 100.0eps. Default value = 0.001eps degree. Default value = 3degree gamma. Default value = 0.01gamma coef0. Default value = 0.0coef0 nu. Default value = 0.5nu p. Default value = 1.0р shrinking shrinking. Default value = 0

number

#### Value

seed

A data.frame with the actual and predicted values for both train and test datasets.

Seed for random numbers. If it is not assigned a value, the seed will be a random

56 Falco\_GP\_C

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::EPSILON_SVR_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Falco\_GP\_C

Falco\_GP\_C KEEL Classification Algorithm

## **Description**

Falco\_GP\_C Classification Algorithm from KEEL.

#### Usage

```
Falco_GP_C(train, test, population_size, max_generations,
   max_deriv_size, rec_prob, mut_prob, copy_prob, alpha, seed)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object

population\_size

population\_size. Default value = 200

max\_generations

max\_generations. Default value = 200

max\_deriv\_size max\_deriv\_size. Default value = 20

rec\_prob rec\_prob. Default value = 0.8
mut\_prob mut\_prob. Default value = 0.1
copy\_prob copy\_prob. Default value = 0.01
alpha alpha. Default value = 0.9

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

FCRA\_C 57

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Falco_GP_C(data_train, data_test)
algorithm <- RKEEL::Falco_GP_C(data_train, data_test, population_size = 5, max_generations = 10)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FCRA\_C

FCRA\_C KEEL Classification Algorithm

### **Description**

FCRA\_C Classification Algorithm from KEEL.

#### Usage

```
FCRA_C(train, test, generations, pop_size, length_S_C, WCAR,
    WV, crossover_prob, mut_prob, n1, n2, max_iter,
    linguistic_values, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
generations generations. Default value = 50
pop\_size pop\_size. Default value = 30
length\_S\_C length\_S\_C. Default value = 10
WCAR WCAR. Default value = 10.0
WV Default value = 1.0

 $\begin{tabular}{llll} $crossover\_prob.$ Default value = 1.0 \\ $mut\_prob.$ Default value = 0.01 \\ $n1$ & $n1.$ Default value = 0.001 \\ $n2$ & $n2.$ Default value = 0.1 \\ $max\_iter$ & $max\_iter.$ Default value = 100 \\ \end{tabular}$ 

linguistic\_values

linguistic\_values. Default value = 5

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::FCRA_C(data_train, data_test, generations=10, pop_size=10)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FPgrowth\_A

FPgrowth\_A KEEL Association Rules Algorithm

### Description

FPgrowth\_A Association Rules Algorithm from KEEL.

#### Usage

```
FPgrowth_A(dat, NumberofPartitionsforNumericAttributes, MinimumSupport,
    MinimumConfidence)
```

# Arguments

```
dat Dataset as a data.frame object

NumberofPartitionsforNumericAttributes

Number of Partitions for Numeric Attributes. Default value = 4

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

MinimumConfidence. Default value = 0.8
```

#### **Details**

```
$run() Run algorith
$showRules(numRules) Show a number of rules. By default all rules.
$getInterestMeasures() Return a data.frame with all interest measures of set rules.
```

FPgrowth\_A 59

```
$sortBy(interestMeasure) Order set rules by interest measure.
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
```

FPgrowth\_A

```
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
```

 $FRNN_C$  61

```
"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)
```

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

#### **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::FPgrowth_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

FRNN\_C

FRNN\_C KEEL Classification Algorithm

## Description

FRNN\_C Classification Algorithm from KEEL.

62 FRSBM\_R

## Usage

```
FRNN_C(train, test)
```

#### **Arguments**

Train dataset as a data.frame object train test Test dataset as a data.frame object

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")</pre>
data_test <- RKEEL::loadKeelDataset("iris_test")</pre>
#Create algorithm
algorithm <- RKEEL::FRNN_C(data_train, data_test)</pre>
#Run algorithm
algorithm$run()
#See results
algorithm$testPredictions
```

FRSBM\_R

FRSBM\_R KEEL Regression Algorithm

## **Description**

FRSBM\_R Regression Algorithm from KEEL.

## Usage

```
FRSBM_R(train, test, numrules, sigma, seed)
```

## **Arguments**

sigma

train Train dataset as a data.frame object Test dataset as a data.frame object test numrules. Default value = 1numrules sigma. Default value = 0.0001

Seed for random numbers. If it is not assigned a value, the seed will be a random seed

number

FURIA\_C 63

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::FRSBM_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FURIA\_C

FURIA\_C KEEL Classification Algorithm

## Description

FURIA\_C Classification Algorithm from KEEL.

## Usage

```
FURIA_C(train, test, optimizations, folds, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object optimizations optimizations. Default value = 2

folds folds. Default value = 3

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

64 FuzzyApriori\_A

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::FURIA_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FuzzyApriori\_A

FuzzyApriori\_A KEEL Association Rules Algorithm

## Description

FuzzyApriori\_A Association Rules Algorithm from KEEL.

## Usage

```
FuzzyApriori_A(dat, NumberofPartitionsforNumericAttributes,
   UseMaxOperatorfor1FrequentItemsets, MinimumSupport, MinimumConfidence)
```

## **Arguments**

dat Dataset as a data.frame object NumberofPartitionsforNumericAttributes

Number of Partitions for Numeric Attributes. Default value = 4

 $Use {\tt MaxOperator} for {\tt 1FrequentItemsets}$ 

Use Max Operator for 1 Frequent Itemsets. Default value = "false"

Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

### **Details**

```
$run() Run algorith
```

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules" \$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported: "allConfidence" (Omiencinski, 2003) "crossSupportRatio", cross-support ratio (Xiong et al., 2003) "lift", interest factor (Brin et al. 1997) "support", supp (Agrawal et al., 1996) "addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002) "chiSquared", X^2 (Liu et al., 1999) "certainty", certainty factor, CF, Loevinger (Berzal et al., 2002) "collectiveStrength" "confidence", conf (Agrawal et al., 1996) "conviction" (Brin et al. 1997) "cosine" (Tan et al., 2004) "coverage", cover, LHS-support "confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999) "casualConfidence", casual confidence (Kodratoff, 1999) "casualSupport", casual support (Kodratoff, 1999) "counterexample", example and counterexample rate "descriptiveConfirm", descriptive-confirm (Kodratoff, 1999) "doc", difference of confidence (Hofmann and Wilhelm, 2001) "fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007) "gini", Gini index (Tan et al., 2004) "hyperLift" (Hahsler and Hornik, 2007)

66 FuzzyApriori\_A

```
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
```

FuzzyFARCHD\_C 67

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

## **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::FuzzyApriori_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

FuzzyFARCHD\_C

FuzzyFARCHD\_C KEEL Classification Algorithm

## Description

FuzzyFARCHD\_C Classification Algorithm from KEEL.

# Usage

```
FuzzyFARCHD_C(train, test, linguistic_values, min_support,
   max_confidence, depth_max, K, max_evaluations, pop_size,
   alpha, bits_per_gen, inference_type, seed)
```

68 FuzzyKNN\_C

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

linguistic\_values

linguistic\_values. Default value = 5

 $min\_support$   $min\_support$ . Default value = 0.05

 $max\_confidence$   $max\_confidence$ . Default value = 0.8

depth\_max. Default value = 3

K. Default value = 2

max\_evaluations

 $max_evaluations$ . Default value = 15000

pop\_size pop\_size. Default value = 50 alpha alpha. Default value = 0.15

bits\_per\_gen bits\_per\_gen. Default value = 30 inference\_type inference\_type. Default value = 1

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::FuzzyFARCHD_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FuzzyKNN\_C

FuzzyKNN\_C KEEL Classification Algorithm

## **Description**

FuzzyKNN\_C Classification Algorithm from KEEL.

FuzzyNPC\_C 69

#### Usage

```
FuzzyKNN_C(train, test, k, M, initialization, init_k)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k k. Default value = 3M Default value = 2.0

initialization initialization. Default value = "CRISP"

init\_k. Default value = 3

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::FuzzyKNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

FuzzyNPC\_C

FuzzyNPC\_C KEEL Classification Algorithm

#### **Description**

FuzzyNPC\_C Classification Algorithm from KEEL.

## Usage

```
FuzzyNPC_C(train, test, M)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object

M. Default value = 2.0

 $GANN_{\_C}$ 

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::FuzzyNPC_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

GANN\_C

GANN\_C KEEL Classification Algorithm

#### **Description**

GANN\_C Classification Algorithm from KEEL.

#### Usage

```
GANN_C(train, test, hidden_layers, hidden_nodes, transfer, eta,
   alpha, lambda, test_data, validation_data, cross_validation,
   BP_cycles, improve, tipify_inputs, save_all, elite,
   num_individuals, w_range, connectivity, P_bp, P_param,
   P_struct, max_generations, seed)
```

## Arguments

Train dataset as a data.frame object train Test dataset as a data.frame object test hidden\_layers. Default value = 2 hidden\_layers hidden\_nodes hidden\_nodes. Default value = 15 transfer transfer. Default value = "Htan" eta. Default value = 0.15eta alpha alpha. Default value = 0.1lambda lambda. Default value = 0.0test\_data test data. Default value = TRUE validation\_data validation\_data. Default value = FALSE *GANN\_C* 71

cross\_validation

cross\_validation. Default value = FALSE

BP\_cycles BP\_cycles. Default value = 10000

improve improve. Default value = 0.01

tipify\_inputs tipify\_inputs. Default value = TRUE

save\_all save\_all. Default value = FALSE

elite elite. Default value = 0.1

num\_individuals

num\_individuals. Default value = 100

w\_range w\_range. Default value = 5.0

connectivity connectivity. Default value = 0.5

P\_bp. Default value = 0.25

P\_param. Default value = 0.1

max\_generations

max\_generations. Default value = 100

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::GANN_C(data_train, data_test, hidden_layers=1, hidden_nodes=5, max_generations=5)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $GAR\_A$ 

GAR\_A

### GAR\_A KEEL Association Rules Algorithm

#### **Description**

GAR\_A Association Rules Algorithm from KEEL.

#### Usage

GAR\_A(dat, seed, NumberofItemsets, TotalNumberofEvaluations, PopulationSize, ProbabilityofSelection, ProbabilityofCrossover, ProbabilityofMutation, ImportanceofNumberofRecordsAlreadyCovered, ImportanceofIntervalsAmplitude, ImportanceofNumberofInvolvedAttributes, AmplitudeFactor, MinimumSupport, MinimumConfidence)

### **Arguments**

dat Dataset as a data.frame object

seed seed. Default value = 1286082570

NumberofItemsets

Number of Itemsets. Default value = 100

TotalNumberofEvaluations

Total Number of Evaluations. Default value = 50000

PopulationSize Population Size. Default value = 100

ProbabilityofSelection

Probability of Selection. Default value = 0.25

ProbabilityofCrossover

Probability of Crossover. Default value = 0.7

Probability of Mutation

Probability of Mutation. Default value = 0.1

 $Importance of {\tt Number of Records Already Covered}$ 

Importance of Number of Records Already Covered. Default value = 0.4

ImportanceofIntervalsAmplitude

Importance of Intervals Amplitude. Default value = 0.7

 $Importance of {\tt Number of Involved Attributes}$ 

Importance of Number of Involved Attributes. Default value = 0.5

AmplitudeFactor

Amplitude Factor. Default value = 2.0

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

 $GAR\_A$ 

#### **Details**

```
$run() Run algorith
$showRules(numRules) Show a number of rules. By default all rules.
$getInterestMeasures() Return a data.frame with all interest measures of set rules.
$sortBy(interestMeasure) Order set rules by interest measure.
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
```

 $GAR\_A$ 

```
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
```

 $GAR\_A$  75

```
"sebag", Sebag measure (Sebag and Schoenauer, 1988)

"support", supp (Agrawal et al., 1996)

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)

"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)

For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("glass")</pre>
#Create algorithm
algorithm <- RKEEL::GAR_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

 $GENAR\_A$ 

GENAR\_A

### GENAR\_A KEEL Association Rules Algorithm

#### **Description**

GENAR\_A Association Rules Algorithm from KEEL.

### Usage

```
GENAR_A(dat, seed, NumberofAssociationRules, TotalNumberofEvaluations,
PopulationSize, ProbabilityofSelection, ProbabilityofMutation,
PenalizationFactor, AmplitudeFactor)
```

### **Arguments**

dat Dataset as a data.frame object

seed. Default value = 1286082570

NumberofAssociationRules

Number of Association Rules. Default value = 30

TotalNumberofEvaluations

Total Number of Evaluations. Default value = 50000

PopulationSize Population Size. Default value = 100

ProbabilityofSelection

Probability of Selection. Default value = 0.25

ProbabilityofMutation

Probability of Mutation. Default value = 0.1

PenalizationFactor

Penalization Factor. Default value = 0.7

AmplitudeFactor

Amplitude Factor. Default value = 2.0

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

GENAR\_A

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

```
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
```

 $GENAR\_A$ 

```
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"iMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

#### **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("glass")</pre>
#Create algorithm
algorithm <- RKEEL::GENAR_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

GeneticFuzzyAprioriDC\_A

 $GeneticFuzzy Apriori DC\_A\ KEEL\ Association\ Rules\ Algorithm$ 

#### **Description**

GeneticFuzzyAprioriDC\_A Association Rules Algorithm from KEEL.

#### Usage

GeneticFuzzyAprioriDC\_A(dat, seed, NumberofEvaluations, PopulationSize, ProbabilityofMutation, ProbabilityofCrossover, ParameterdforMMACrossover, NumberofFuzzyRegionsforNumericAttributes, UseMaxOperatorfor1FrequentItemsets, MinimumSupport, MinimumConfidence)

#### **Arguments**

dat Dataset as a data.frame object

seed seed. Default value = 1286082570

NumberofEvaluations

Number of Evaluations. Default value = 10000

PopulationSize Population Size. Default value = 50

ProbabilityofMutation

Probability of Mutation. Default value = 0.01

ProbabilityofCrossover

Probability of Crossover. Default value = 0.8

ParameterdforMMACrossover

Parameterd for MMA Crossover. Default value = 0.35

NumberofFuzzyRegionsforNumericAttributes

Number of Fuzzy Regions for Numeric Attributes. Default value = 3

UseMaxOperatorfor1FrequentItemsets

Use Max Operator for 1 Frequent Itemsets. Default value = "false"

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

"allConfidence" (Omiencinski, 2003)

"crossSupportRatio", cross-support ratio (Xiong et al., 2003)

"lift", interest factor (Brin et al. 1997)

"support", supp (Agrawal et al., 1996)

```
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
```

```
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")

#Create algorithm
algorithm <- RKEEL::GeneticFuzzyAprioriDC_A(dat)</pre>
```

```
#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

#Show a number of rules
algorithm$showRules(2)

#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()

#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")

#Sort by interest measure lift
algorithm$sortBy("lift")

#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

GeneticFuzzyApriori\_A KEEL Association Rules Algorithm

#### **Description**

GeneticFuzzyApriori\_A Association Rules Algorithm from KEEL.

#### Usage

```
GeneticFuzzyApriori_A(dat, seed, NumberofEvaluations, PopulationSize, ProbabilityofMutation, ProbabilityofCrossover, ParameterdforMMACrossover, NumberofFuzzyRegionsforNumericAttributes, UseMaxOperatorfor1FrequentItemsets, MinimumSupport, MinimumConfidence)
```

# **Arguments**

dat Dataset as a data.frame object
seed seed. Default value = 1286082570
NumberofEvaluations
Number of Evaluations. Default value = 10000
PopulationSize Population Size. Default value = 50
ProbabilityofMutation
Probability of Mutation. Default value = 0.01
ProbabilityofCrossover

Probability of Crossover. Default value = 0.8

```
ParameterdforMMACrossover
```

Parameterd for MMA Crossover. Default value = 0.35

 ${\tt Number of Fuzzy Regions for Numeric Attributes}$ 

Number of Fuzzy Regions for Numeric Attributes. Default value = 3

UseMaxOperatorfor1FrequentItemsets

Use Max Operator for 1 Frequent Itemsets. Default value = "false"

MinimumSupport Minimum Support. Default value = 0.1

MinimumConfidence

Minimum Confidence. Default value = 0.8

#### **Details**

```
$run() Run algorith
```

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

```
"allConfidence" (Omiencinski, 2003)
```

"lift", interest factor (Brin et al. 1997)

"support", supp (Agrawal et al., 1996)

"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)

"chiSquared", X^2 (Liu et al., 1999)

"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)

"collectiveStrength"

"confidence", conf (Agrawal et al., 1996)

"conviction" (Brin et al. 1997)

<sup>&</sup>quot;crossSupportRatio", cross-support ratio (Xiong et al., 2003)

```
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
```

```
"leverage", PS (Piatetsky-Shapiro 1991)

"mutualInformation", uncertainty, M (Tan et al., 2002)

"oddsRatio", odds ratio alpha (Tan et al., 2004)

"phi", correlation coefficient phi (Tan et al. 2004)

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)

"sebag", Sebag measure (Sebag and Schoenauer, 1988)

"support", supp (Agrawal et al., 1996)

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)

"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)

For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")

#Create algorithm
algorithm <- RKEEL::GeneticFuzzyApriori_A(dat)

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

#Show a number of rules
algorithm$showRules(2)

#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()</pre>
```

```
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY","yulesY")

#Sort by interest measure lift
algorithm$sortBy("lift")

#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

getAttributeLinesFromDataframes

Get attribute lines from data.frames

# Description

Method for getting the attribute lines from data.frame objects

#### Usage

```
getAttributeLinesFromDataframes(trainData, testData)
```

### **Arguments**

trainData Train dataset as data.frame
testData Test dataset as data.frame

#### Value

Returns a list with the attribute names and types

```
iris_train <- RKEEL::loadKeelDataset("iris_train")
iris_test <- RKEEL::loadKeelDataset("iris_test")
attributeLines <- getAttributeLinesFromDataframes(iris_train, iris_test)</pre>
```

88 getJarList

getExePath

Get jar executable files Path

# Description

Method for knowing the KEEL .jar files path.

# Usage

```
getExePath()
```

#### Value

Returns a string with the path of the KEEL .jar files.

# **Examples**

```
getExePath()
```

getJarList

Get a list with all RKEEL algorithm jars

# Description

Method that returns a list with the jar names from RKEEL

# Usage

```
getJarList()
```

### Value

Returns a list with the jar names from RKEEL.

```
getJarList()
```

getJarPath 89

getJarPath	Get RunKeel.jar Path

# Description

Method for knowing the RunKeel.jar path.

#### Usage

```
getJarPath()
```

#### Value

Returns a string with the RunKeel.jar path.

# **Examples**

```
getJarPath()
```

GFS\_AdaBoost\_C

GFS\_AdaBoost\_C KEEL Classification Algorithm

### **Description**

GFS\_AdaBoost\_C Classification Algorithm from KEEL.

# Usage

```
GFS_AdaBoost_C(train, test, numLabels, numRules, seed)
```

# **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numLabels numLabels. Default value = 3
numRules numRules. Default value = 8

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

90 GFS\_GP\_R

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::GFS_AdaBoost_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

GFS\_GP\_R

GFS\_GP\_R KEEL Regression Algorithm

### **Description**

GFS\_GP\_R Regression Algorithm from KEEL.

# Usage

```
GFS_GP_R(train, test, numLabels, numRules, popSize, numisland,
   steady, numIter, tourSize, mutProb, aplMut, probMigra,
   probOptimLocal, numOptimLocal, idOptimLocal, nichinggap,
   maxindniche, probintraniche, probcrossga, probmutaga,
   lenchaingap, maxtreeheight, seed)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object numLabels numLabels. Default value = 3 numRules. Default value = 8 numRules popSize. Default value = 30popSize numisland numisland. Default value = 2steady steady. Default value = 1numIter. Default value = 100 numIter tourSize. Default value = 4tourSize mutProb mutProb. Default value = 0.01aplMut aplMut. Default value = 0.1probMigra probMigra. Default value = 0.001 probOptimLocal probOptimLocal. Default value = 0.00 GFS\_GSP\_R 91

numOptimLocal. Default value = 0 numOptimLocal idOptimLocal idOptimLocal. Default value = 0 nichinggap nichinggap. Default value = 0maxindniche maxindniche. Default value = 8 probintraniche probintraniche. Default value = 0.75 probcrossga probcrossga. Default value = 0.5probmutaga probmutaga. Default value = 0.5lenchaingap. Default value = 10 lenchaingap maxtreeheight maxtreeheight. Default value = 8 seed Seed for random numbers. If it is not assigned a value, the seed will be a random number

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::GFS_GP_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

GFS\_GSP\_R

GFS\_GSP\_R KEEL Regression Algorithm

### **Description**

GFS\_GSP\_R Regression Algorithm from KEEL.

#### Usage

```
GFS_GSP_R(train, test, numLabels, numRules, deltafitsap,
   p0sap, p1sap, amplMut, nsubsap, probOptimLocal,
   numOptimLocal, idOptimLocal, probcrossga, probmutaga,
   lenchaingap, maxtreeheight, numItera, seed)
```

92 GFS\_GSP\_R

# Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object numLabels numLabels. Default value = 3numRules numRules. Default value = 8 deltafitsap deltafitsap. Default value = 0.5p0sap. Default value = 0.5p0sap p1sap p1sap. Default value = 0.5amplMut. Default value = 0.1amplMut nsubsap. Default value = 10 nsubsap probOptimLocal probOptimLocal. Default value = 0.00 numOptimLocal numOptimLocal. Default value = 0 idOptimLocal. Default value = 0 idOptimLocal probcrossga probcrossga. Default value = 0.5probmutaga. Default value = 0.5probmutaga lenchaingap lenchaingap. Default value = 10 maxtreeheight. Default value = 8 maxtreeheight numItera. Default value = 10000 numItera seed Seed for random numbers. If it is not assigned a value, the seed will be a random

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

number

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::GFS_GSP_R(data_train, data_test)
algorithm <- RKEEL::GFS_GSP_R(data_train, data_test, numRules=2, numItera=10, maxtreeheight=2)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

GFS\_LogitBoost\_C 93

GFS\_LogitBoost\_C

GFS\_LogitBoost\_C KEEL Classification Algorithm

### **Description**

GFS\_LogitBoost\_C Classification Algorithm from KEEL.

# Usage

```
GFS_LogitBoost_C(train, test, numLabels, numRules, seed)
```

### **Arguments**

train Train dataset as a data.frame object

test Test dataset as a data.frame object

numLabels numLabels. Default value = 3

numRules numRules. Default value = 25

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::GFS_LogitBoost_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

94 GFS\_RB\_MF\_R

GFS\_RB\_MF\_R

GFS\_RB\_MF\_R KEEL Regression Algorithm

### **Description**

GFS\_RB\_MF\_R Regression Algorithm from KEEL.

#### Usage

```
GFS_RB_MF_R(train, test, numLabels, popSize, generations,
    crossProb, mutProb, seed)
```

#### **Arguments**

train Train dataset as a data.frame object Test dataset as a data.frame object test numLabels numLabels. Default value = 3 popSize. Default value = 50 popSize generations generations. Default value = 100 crossProb. Default value = 0.9 crossProb mutProb mutProb. Default value = 0.1 Seed for random numbers. If it is not assigned a value, the seed will be a random seed number

# Value

A data.frame with the actual and predicted values for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::GFS_RB_MF_R(data_train, data_test)
algorithm <- RKEEL::GFS_RB_MF_R(data_train, data_test, popSize = 5, generations = 10)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

hasContinuousData 95

hasContinuousData

Has Continuous Data

# Description

Method for check if a dataset has continuous data

### Usage

hasContinuousData(data)

# Arguments

data

Dataset as data.frame

#### Value

Returns TRUE if the dataset has continuous data and FALSE if it has not.

# **Examples**

```
iris <- RKEEL::loadKeelDataset("iris")
hasContinuousData(iris)</pre>
```

hasMissingValues

Has Missing Values

### **Description**

Method for check if a dataset has missing values

#### Usage

```
hasMissingValues(data)
```

### Arguments

data

Dataset as data.frame

### Value

Returns TRUE if the dataset has missing values and FALSE if it has not.

```
iris <- RKEEL::loadKeelDataset("iris")
hasMissingValues(iris)</pre>
```

96 ID3\_D

ID3\_C

ID3\_C KEEL Classification Algorithm

# Description

ID3\_C Classification Algorithm from KEEL.

### Usage

```
ID3_C(train, test)
```

### Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ID3_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

ID3\_D

ID3\_D KEEL Preprocess Algorithm

# Description

ID3\_D Preprocess Algorithm from KEEL.

### Usage

```
ID3_D(train, test)
```

*IF\_KNN\_C* 97

# **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

# **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ID3_D(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

IF\_KNN\_C

IF\_KNN\_C KEEL Classification Algorithm

# Description

IF\_KNN\_C Classification Algorithm from KEEL.

### Usage

```
IF_KNN_C(train, test, K, mA, vA, mR, vR, k)
```

#### **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
K	K. Default value = 3
mA	mA. Default value = $0.6$
vA	vA. Default value = $0.4$
mR	mR. Default value = $0.3$
vR	vR. Default value = $0.7$
k	k. Default value = 5

98 Ignore\_MV

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::IF_KNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Ignore\_MV

Ignore\_MV KEEL Preprocess Algorithm

#### **Description**

Ignore\_MV Preprocess Algorithm from KEEL.

#### Usage

```
Ignore_MV(train, test)
```

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::Ignore_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

 ${\bf Imbalanced Classification Algorithm}$ 

ImbalancedClassification Algorithm

### **Description**

Class inheriting of ClassificationAlgorithm, to common methods for all KEEL Imbalanced Classification Algorithms. The specific imbalanced-classification algorithms must inherit of this class.

IncrRBFN\_C

IncrRBFN\_C KEEL Classification Algorithm

### **Description**

IncrRBFN\_C Classification Algorithm from KEEL.

### Usage

```
IncrRBFN_C(train, test, epsilon, alfa, delta, seed)
```

# **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
epsilon	epsilon. Default value = 0.1
alfa	alfa. Default value = $0.3$
delta	delta. Default value = $0.5$
seed	Seed for random numbers. If it is not assigned a va

value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")</pre>
data_test <- RKEEL::loadKeelDataset("iris_test")</pre>
#Create algorithm
algorithm <- RKEEL::IncrRBFN_C(data_train, data_test)</pre>
#Run algorithm
algorithm$run()
#See results
algorithm$testPredictions
```

isMultiClass

Is Multi-class

### Description

Method for check if a dataset is multi-class

#### Usage

```
isMultiClass(data)
```

#### **Arguments**

data

Dataset as data.frame

#### Value

Returns TRUE if the dataset is multi-class and FALSE if it is not.

### **Examples**

```
iris <- RKEEL::loadKeelDataset("iris")
isMultiClass(iris)</pre>
```

IterativePartitioningFilter\_F

IterativePartitioningFilter\_F KEEL Preprocess Algorithm

### **Description**

IterativePartitioningFilter\_F Preprocess Algorithm from KEEL.

### Usage

```
IterativePartitioningFilter_F(train, test, numPartitions,
    filterType, confidence, itemsetsPerLeaf, seed)
```

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numPartitions numPartitions. Default value = 5
filterType filterType. Default value = "consensus"
confidence confidence. Default value = 0.25

 $\verb|itemsetsPerLeaf|$ 

itemsetsPerLeaf. Default value = 2

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

*JFKNN\_C* 101

#### Value

A data.frame with the preprocessed data for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::IterativePartitioningFilter_F(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

JFKNN\_C

JFKNN\_C KEEL Classification Algorithm

### **Description**

JFKNN\_C Classification Algorithm from KEEL.

### Usage

```
JFKNN_C(train, test)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::JFKNN_C(data_train, data_test)

#Run algorithm
algorithm$run()</pre>
```

102 Kernel\_C

```
#See results
algorithm$testPredictions
```

KeelAlgorithm

Keel Algorithm

### **Description**

Principal class for implementing KEEL Algorithms. The distinct types of algorithms must inherit of this class.

Kernel\_C

Kernel\_C KEEL Classification Algorithm

### **Description**

Kernel\_C Classification Algorithm from KEEL.

#### **Usage**

```
Kernel_C(train, test, sigma, seed)
```

### Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
sigma	sigma. Default value = $0.01$

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Kernel_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

KMeans\_MV 103

KMeans_	_MV
---------	-----

KMeans\_MV KEEL Preprocess Algorithm

### **Description**

KMeans\_MV Preprocess Algorithm from KEEL.

### Usage

```
KMeans_MV(train, test, k, error, iterations, seed)
```

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

k. Default value = 10

error. Default value = 100

iterations iterations. Default value = 100

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::KMeans_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

104 KNN\_MV

KNN\_C

KNN-C KEEL Classification Algorithm

#### **Description**

KNN-C Classification Algorithm from KEEL.

### Usage

```
KNN_C(train, test, k, distance)
```

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k Number of neighbors distance Distance function

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::KNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

KNN\_MV

KNN\_MV KEEL Preprocess Algorithm

### **Description**

KNN\_MV Preprocess Algorithm from KEEL.

#### Usage

```
KNN_MV(train, test, k)
```

KSNN\_C 105

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object

k k. Default value = 10

### Value

A data.frame with the preprocessed data for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::KNN_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

KSNN\_C

KSNN\_C KEEL Classification Algorithm

# Description

KSNN\_C Classification Algorithm from KEEL.

### Usage

```
KSNN_C(train, test, k)
```

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

k. Default value = 1

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

106 KStar\_C

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::KSNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

KStar\_C

KStar\_C KEEL Classification Algorithm

# Description

KStar\_C Classification Algorithm from KEEL.

#### Usage

```
KStar_C(train, test, selection_method, blend, seed)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

selection\_method

selection\_method. Default value = "Fixed"

blend blend. Default value = 0.2

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::KStar_C(data_train, data_test)

#Run algorithm</pre>
```

LDA\_C 107

```
algorithm$run()
#See results
algorithm$testPredictions
```

LDA\_C

LDA\_C KEEL Classification Algorithm

# Description

LDA\_C Classification Algorithm from KEEL.

# Usage

```
LDA_C(train, test, seed)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::LDA_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

108 LinearLMS\_R

LinearLMS\_C

LinearLMS\_C KEEL Classification Algorithm

#### **Description**

LinearLMS\_C Classification Algorithm from KEEL.

#### Usage

```
LinearLMS_C(train, test, seed)
```

### Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::LinearLMS_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

LinearLMS\_R

LinearLMS\_R KEEL Regression Algorithm

### **Description**

LinearLMS\_R Regression Algorithm from KEEL.

#### Usage

```
LinearLMS_R(train, test, seed)
```

loadKeelDataset 109

#### **Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

# **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::LinearLMS_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

loadKeelDataset

Load KEEL Dataset

## **Description**

Loads a dataset of the KEEL datasets repository. The included datasets names are available at the getKeelDatasetList method of RKEELdata.

### Usage

```
loadKeelDataset(dataName)
```

#### **Arguments**

dataName

String with the correct data name of one of the KEEL datasets

## Value

Returns a data frame with the KEEL dataset.

```
RKEEL::loadKeelDataset("iris")
```

110 LVF\_IEP\_FS

Logistic\_C

Logistic\_C KEEL Classification Algorithm

### **Description**

Logistic\_C Classification Algorithm from KEEL.

## Usage

```
Logistic_C(train, test, ridge, maxIter)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
ridge ridge. Default value = 1e-8
maxIter maxIter. Default value = -1

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Logistic_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

LVF\_IEP\_FS

LVF\_IEP\_FS KEEL Preprocess Algorithm

## **Description**

LVF\_IEP\_FS Preprocess Algorithm from KEEL.

### Usage

```
LVF_IEP_FS(train, test, paramKNN, maxLoops, inconAllow, seed)
```

M5Rules\_R

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
paramKNN paramKNN. Default value = 1
maxLoops maxLoops. Default value = 770
inconAllow inconAllow. Default value = 0

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::LVF_IEP_FS(data_train, data_test)
algorithm <- RKEEL::LVF_IEP_FS(data_train, data_test, maxLoops = 30, inconAllow=2)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

M5Rules\_R

M5Rules\_R KEEL Regression Algorithm

## Description

M5Rules\_R Regression Algorithm from KEEL.

## Usage

```
M5Rules_R(train, test, pruningFactor, heuristic)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
pruningFactor pruningFactor. Default value = 2
heuristic heuristic. Default value = "Coverage"

112 M5\_R

## Value

A data.frame with the actual and predicted values for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::M5Rules_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

M5\_R

M5\_R KEEL Regression Algorithm

## **Description**

M5\_R Regression Algorithm from KEEL.

#### Usage

```
M5_R(train, test, type, pruningFactor, unsmoothed)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

type type. Default value = "m"

 $\begin{array}{ll} \mbox{pruningFactor} & \mbox{pruningFactor. Default value} = 2 \\ \mbox{unsmoothed} & \mbox{unsmoothed. Default value} = TRUE \end{array}$ 

## Value

A data.frame with the actual and predicted values for both train and test datasets.

MinMax\_TR 113

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::M5_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

MinMax\_TR

MinMax\_TR KEEL Preprocess Algorithm

# **Description**

MinMax\_TR Preprocess Algorithm from KEEL.

### Usage

```
MinMax_TR(train, test, newMin, newMax)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
newMin newMin. Default value = 0.0
newMax Default value = 1.0

## Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::MinMax_TR(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

114 *MLP\_BP\_C* 

MLP\_BP\_C

MLP\_BP\_C KEEL Classification Algorithm

#### **Description**

MLP\_BP\_C Classification Algorithm from KEEL.

## Usage

```
MLP_BP_C(train, test, hidden_layers, hidden_nodes, transfer,
  eta, alpha, lambda, test_data, validation_data,
  cross_validation, cycles, improve, tipify_inputs,
  save_all, seed)
```

#### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
hidden\_layers hidden\_layers. Default value = 2
hidden\_nodes hidden\_nodes. Default value = 15
transfer transfer. Default value = "Htan"

eta eta. Default value = 0.15 alpha alpha. Default value = 0.1 lambda lambda. Default value = 0.0

test\_data test\_data. Default value = TRUE

validation\_data

validation data. Default value = FALSE

cross\_validation

cross\_validation. Default value = FALSE

cycles cycles. Default value = 10000 improve improve. Default value = 0.01

tipify\_inputs tipify\_inputs. Default value = TRUE save\_all save\_all. Default value = FALSE

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

 $MLP\_BP\_R$ 115

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")</pre>
data_test <- RKEEL::loadKeelDataset("iris_test")</pre>
#Create algorithm
algorithm <- RKEEL::MLP_BP_C(data_train, data_test, )</pre>
#Run algorithm
algorithm$run()
#See results
algorithm$testPredictions
```

MLP\_BP\_R

MLP\_BP\_R KEEL Regression Algorithm

## **Description**

MLP\_BP\_R Regression Algorithm from KEEL.

## Usage

```
MLP_BP_R(train, test, hidden_layers, hidden_nodes, transfer,
   eta, alpha, lambda, test_data, validation_data,
   cross_validation, cycles, improve, tipify_inputs,
   save_all, seed)
```

## **Arguments**

train

Train dataset as a data.frame object Test dataset as a data.frame object test hidden\_layers. Default value = 2 hidden\_layers hidden\_nodes hidden\_nodes. Default value = 15 transfer transfer. Default value = "Htan" eta. Default value = 0.15eta alpha. Default value = 0.1alpha lambda lambda. Default value = 0.0test\_data. Default value = TRUE test\_data validation\_data validation\_data. Default value = FALSE cross\_validation cross validation. Default value = FALSE cycles. Default value = 10000 cycles

116 ModelCS\_TSS

improve improve. Default value = 0.01

tipify\_inputs tipify\_inputs. Default value = TRUE save\_all save\_all. Default value = FALSE

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::MLP_BP_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

ModelCS\_TSS

ModelCS\_TSS KEEL Preprocess Algorithm

## **Description**

ModelCS\_TSS Preprocess Algorithm from KEEL.

## Usage

```
ModelCS_TSS(train, test, k, distance)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k. Default value = 3

distance distance. Default value = "Euclidean"

### Value

A data.frame with the preprocessed data for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ModelCS_TSS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

MODENAR\_A

MODENAR\_A KEEL Association Rules Algorithm

## **Description**

MODENAR\_A Association Rules Algorithm from KEEL.

# Usage

```
MODENAR_A(dat, seed, PopulationSize, NumberofEvaluations, CrossoverrateCR, Thresholdforthenumberofnondominatedsolutions, Thefactorofamplitudeforeachattributeofthedataset, WeightforSupport, WeightforConfidence, WeightforComprehensibility, WeightforAmplitudeoftheIntervals)
```

#### **Arguments**

dat Dataset as a data.frame object
seed seed. Default value = 1286082570
PopulationSize Population Size. Default value = 100
NumberofEvaluations

Number of Evaluations. Default value = 50000

CrossoverrateCR

Crossover rate CR. Default value = 0.3

Thresholdforthenumberofnondominatedsolutions

Threshold for the number of non-dominated solutions. Default value = 60

The factor of amplitude for each attribute of the dataset

The factor of amplitude for each attribute of the dataset. Default value = 2

WeightforSupport

Weight for Support. Default value = 0.8

WeightforConfidence

Weight for Confidence. Default value = 0.2

```
WeightforComprehensibility Weight for Comprehensibility. Default value = 0.1 WeightforAmplitudeoftheIntervals Weight for Amplitude of the Intervals. Default value = 0.4
```

#### **Details**

```
$run() Run algorith
$showRules(numRules) Show a number of rules. By default all rules.
$getInterestMeasures() Return a data.frame with all interest measures of set rules.
$sortBy(interestMeasure) Order set rules by interest measure.
$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
```

```
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
```

```
"phi", correlation coefficient phi (Tan et al. 2004)

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)

"sebag", Sebag measure (Sebag and Schoenauer, 1988)

"support", supp (Agrawal et al., 1996)

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)

"yuleQ", Yule's Q (Tan and Kumar, 2000)

"yuleY", Yule's Y (Tan and Kumar, 2000)

For more information see ?arules::interestMeasure
```

#### Value

A arules class with the Association Rules for both dat dataset.

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::MODENAR_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
```

MOEA\_Ghosh\_A

```
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

MOEA\_Ghosh\_A

MOEA\_Ghosh\_A KEEL Association Rules Algorithm

## **Description**

MOEA\_Ghosh\_A Association Rules Algorithm from KEEL.

## Usage

```
MOEA_Ghosh_A(dat, seed, NumberofObjetives, NumberofEvaluations, PopulationSize, PointCrossover, ProbabilityofCrossover, ProbabilityofMutation, Thefactorofamplitudeforeachattributeofthedataset)
```

#### **Arguments**

dat Dataset as a data.frame object seed seed. Default value = 1286082570

NumberofObjetives

Number of Objetives. Default value = 3

NumberofEvaluations

Number of Evaluations. Default value = 50000

PopulationSize Population Size. Default value = 100 PointCrossover Point Crossover. Default value = 2

ProbabilityofCrossover

Probability of Crossover. Default value = 0.8

ProbabilityofMutation

Probability of Mutation. Default value = 0.02

The factor of amplitude for each attribute of the dataset

The factor of amplitude for each attribute of the dataset. Default value = 2.0

### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

122 MOEA\_Ghosh\_A

```
$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"
$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest mea-
sures supported:
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
```

MOEA\_Ghosh\_A 123

```
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
```

For more information see ?arules::interestMeasure

#### Value

A arules class with the Association Rules for both dat dataset.

# **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::MOEA_Ghosh_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

MOPNAR\_A

MOPNAR\_A KEEL Association Rules Algorithm

# Description

MOPNAR\_A Association Rules Algorithm from KEEL.

#### Usage

```
MOPNAR_A(dat, seed, objetives, evaluations, parameter, weightNeighborhood, wrobabilitySolutionsNeighborhood, maxSolutions, probabilityMutation, amplitude, threshold)
```

## **Arguments**

dat Dataset as a data.frame object seed seed. Default value = 1286082570

objetives objetives. Default value = 3

evaluations evaluations. Default value = 50000

parameter. Default value = 13

 ${\tt weightNeighborhood}$ 

weightNeighborhood. Default value = 10

wrobabilitySolutionsNeighborhood

wrobabilitySolutionsNeighborhood. Default value = 0.9

maxSolutions maxSolutions. Default value = 2

probabilityMutation

probabilityMutation. Default value = 0.1

amplitude amplitude. Default value = 2.0 threshold threshold. Default value = 5.0

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

"allConfidence" (Omiencinski, 2003)

"crossSupportRatio", cross-support ratio (Xiong et al., 2003)

"lift", interest factor (Brin et al. 1997)

```
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
```

```
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

## Value

A arules class with the Association Rules for both dat dataset.

## **Examples**

#Load KEEL dataset

128 MostCommon\_MV

```
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::MOPNAR_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

 ${\tt MostCommon\_MV}$ 

MostCommon\_MV KEEL Preprocess Algorithm

# Description

MostCommon\_MV Preprocess Algorithm from KEEL.

### Usage

```
MostCommon_MV(train, test)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

*NB\_C* 129

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::MostCommon_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

NB\_C

NB\_C KEEL Classification Algorithm

#### **Description**

NB\_C Classification Algorithm from KEEL.

### Usage

```
NB_C(train, test)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::NB_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

NICGAR\_A

NICGAR\_A

NICGAR\_A KEEL Association Rules Algorithm

#### **Description**

NICGAR\_A Association Rules Algorithm from KEEL.

### Usage

```
NICGAR_A(dat, seed, NumberofEvaluations, PopulationSize, ProbabilityofMutation, Thefactorofamplitudeforeachattributeofthedataset, NichingThreshold, QualityThreshold, PercentUpdate)
```

## Arguments

dat Dataset as a data.frame object seed Seed. Default value = 1286082570

NumberofEvaluations

Number of Evaluations. Default value = 1286082570

PopulationSize Population Size. Default value = 1286082570

ProbabilityofMutation

Probability of Mutation. Default value = 1286082570

The factor of amplitude for each attribute of the dataset

The factor of amplitude for each attribute of the dataset. Default value = 1286082570

NichingThreshold

Niching Threshold. Default value = 1286082570

QualityThreshold

Quality Threshold. Default value = 1286082570

PercentUpdate Percent Update. Default value = 1286082570

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

NICGAR\_A 131

```
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
```

NICGAR\_A

```
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"jMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

*NM\_C* 133

## Value

A arules class with the Association Rules for both dat dataset.

### **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::NICGAR_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

NM\_C

NM\_C KEEL Classification Algorithm

# Description

NM\_C Classification Algorithm from KEEL.

# Usage

```
NM_C(train, test)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

NNEP\_C

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::NM_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

NNEP\_C

NNEP\_C KEEL Classification Algorithm

## **Description**

NNEP\_C Classification Algorithm from KEEL.

## Usage

```
NNEP_C(train, test, hidden_nodes, transfer, generations, seed)
```

## Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
hidden\_nodes hidden\_nodes. Default value = 4

transfer transfer. Default value = "Product\_Unit"

generations generations. Default value = 200

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

Nominal2Binary\_TR 135

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::NNEP_C(data_train, data_test)
algorithm <- RKEEL::NNEP_C(data_train, data_test, generations = 5)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Nominal2Binary\_TR

Nominal2Binary\_TR KEEL Preprocess Algorithm

## **Description**

Nominal2Binary\_TR Preprocess Algorithm from KEEL.

## Usage

```
Nominal2Binary_TR(train, test)
```

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

## Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::Nominal2Binary_TR(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

136 *NU\_SVM\_C* 

NU\_SVM\_C

NU\_SVM\_C KEEL Classification Algorithm

#### **Description**

NU\_SVM\_C Classification Algorithm from KEEL.

## Usage

```
NU_SVM_C(train, test, KernelType, C, eps, degree, gamma, coef0,
    nu, p, shrinking, seed)
```

## **Arguments**

Train dataset as a data.frame object train Test dataset as a data.frame object test KernelType KernelType. Default value = 1C. Default value = "RBF" С eps. Default value = 1000.0eps degree. Default value = 0.001degree gamma. Default value = 10 gamma coef0. Default value = 0.01coef0 nu. Default value = 0.1nu p. Default value = 1.0shrinking shrinking. Default value = 1

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::NU_SVM_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

*NU\_SVR\_R* 137

NU\_SVR\_R

NU\_SVR\_R KEEL Regression Algorithm

#### **Description**

NU\_SVR\_R Regression Algorithm from KEEL.

## Usage

```
NU_SVR_R(train, test, KernelType, C, eps, degree, gamma,
coef0, nu, p, shrinking, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
KernelType KernelType. Default value = ?

C C. Default value = ?
eps eps. Default value = ?
degree degree. Default value = ?
gamma gamma. Default value = ?
coef0 coef0. Default value = ?
nu nu. Default value = ?
p. Default value = ?

shrinking shrinking. Default value = ?

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

## Value

A data.frame with the actual and predicted values for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::NU_SVR_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

138 PDFC\_C

PART\_C

PART\_C KEEL Classification Algorithm

## **Description**

PART\_C Classification Algorithm from KEEL.

# Usage

```
PART_C(train, test, confidence, itemsetsPerLeaf)
```

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object confidence confidence. Default value = 0.25itemsetsPerLeaf

itemsetsPerLeaf. Default value = 2

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")</pre>
data_test <- RKEEL::loadKeelDataset("iris_test")</pre>
#Create algorithm
algorithm <- RKEEL::PART_C(data_train, data_test)</pre>
#Run algorithm
algorithm$run()
#See results
algorithm$testPredictions
```

PDFC\_C

PDFC\_C KEEL Classification Algorithm

## **Description**

PDFC\_C Classification Algorithm from KEEL.

PDFC\_C 139

## Usage

# Arguments

train	Train dataset as a data.frame object	
test	Test dataset as a data.frame object	
С	C. Default value = 100.0	
d	d. Default value = 0.25	
tolerance	tolerance. Default value = 0.001	
epsilon	epsilon. Default value = 1.0E-12	
PDRFtype	PDRFtype. Default value = "Gaussian	
nominal_to_binary		
	nominal_to_binary. Default value = TRUE	
preprocess_type		
	<pre>preprocess_type. Default value = "Normalize"</pre>	
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number	

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PDFC_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

PNN\_C

PFKNN\_C

PFKNN\_C KEEL Classification Algorithm

## **Description**

PFKNN\_C Classification Algorithm from KEEL.

## Usage

```
PFKNN_C(train, test, k, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

k. Default value = 3

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PFKNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

PNN\_C

PNN\_C KEEL Classification Algorithm

## Description

PNN\_C Classification Algorithm from KEEL.

PolQuadraticLMS\_C 141

#### Usage

```
PNN_C(train, test, seed)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

PolQuadraticLMS\_C

PolQuadraticLMS\_C KEEL Classification Algorithm

## **Description**

PolQuadraticLMS\_C Classification Algorithm from KEEL.

### Usage

```
PolQuadraticLMS_C(train, test, seed)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

142 PolQuadraticLMS\_R

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PolQuadraticLMS_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

PolQuadraticLMS\_R

PolQuadraticLMS\_R KEEL Regression Algorithm

## **Description**

PolQuadraticLMS\_R Regression Algorithm from KEEL.

## Usage

```
PolQuadraticLMS_R(train, test, seed)
```

### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::PolQuadraticLMS_R(data_train, data_test)

#Run algorithm</pre>
```

POP\_TSS 143

```
algorithm$run()
#See results
algorithm$testPredictions
```

POP\_TSS

POP\_TSS KEEL Preprocess Algorithm

# Description

POP\_TSS Preprocess Algorithm from KEEL.

# Usage

```
POP_TSS(train, test)
```

# Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

## Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::POP_TSS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

PRISM\_C

Preprocess Algorithm Preprocess Algorithm

### **Description**

Class inheriting of KeelAlgorithm, to common methods for all KEEL Preprocess Algorithms. The specific preprocessing algorithms must inherit of this class.

The run() method receives three parameters. The folderPath parameter indicates where to place the folder with the experiments if wanted. If it is not indicated, the folder is placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

PRISM\_C

PRISM\_C KEEL Classification Algorithm

#### **Description**

PRISM\_C Classification Algorithm from KEEL.

#### Usage

```
PRISM_C(train, test, seed)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::PRISM_C(data_train, data_test)</pre>
```

Proportional\_D 145

```
#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

Proportional\_D

Proportional\_D KEEL Preprocess Algorithm

#### **Description**

Proportional\_D Preprocess Algorithm from KEEL.

#### Usage

```
Proportional_D(train, test, seed)
```

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

## Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::Proportional_D(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

146 PSO\_ACO\_C

PSO\_ACO\_C

PSO\_ACO\_C KEEL Classification Algorithm

#### **Description**

PSO\_ACO\_C Classification Algorithm from KEEL.

#### Usage

```
PSO_ACO_C(train, test, max_uncovered_samples, min_saples_by_rule,
    max_iterations_without_converge, environmentSize, numParticles,
    x, c1, c2, seed)
```

#### **Arguments**

```
train
                 Train dataset as a data.frame object
                 Test dataset as a data.frame object
test
max_uncovered_samples
                 max_uncovered_samples. Default value = 20
min_saples_by_rule
                 min_saples_by_rule. Default value = 2
max_iterations_without_converge
                  max_iterations_without_converge. Default value = 100
enviromentSize enviromentSize. Default value = 3
                  numParticles. Default value = 100
numParticles
                  x. Default value = 0.72984
Χ
                 c1. Default value = 2.05
c1
                  c2. Default value = 2.05
c2
                  Seed for random numbers. If it is not assigned a value, the seed will be a random
seed
                  number
```

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PSO_ACO_C(data_train, data_test,
    max_iterations_without_converge=2, numParticles=5)</pre>
```

PSRCG\_TSS 147

```
#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

PSRCG\_TSS

PSRCG\_TSS KEEL Preprocess Algorithm

## **Description**

PSRCG\_TSS Preprocess Algorithm from KEEL.

## Usage

```
PSRCG_TSS(train, test, distance)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
distance distance. Default value = "Euclidean"

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::PSRCG_TSS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

148 PUBLIC\_C

PUBLIC\_C

PUBLIC\_C KEEL Classification Algorithm

## **Description**

PUBLIC\_C Classification Algorithm from KEEL.

## Usage

```
PUBLIC_C(train, test, nodesBetweenPrune, estimateToPrune)
```

## **Arguments**

```
train Train dataset as a data.frame object

test Test dataset as a data.frame object

nodesBetweenPrune
nodesBetweenPrune. Default value = 25

estimateToPrune
estimateToPrune. Default value = "PUBLIC(1)"
```

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PUBLIC_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

PW\_C 149

PW\_C

PW\_C KEEL Classification Algorithm

# Description

PW\_C Classification Algorithm from KEEL.

# Usage

```
PW_C(train, test, beta, ro, epsilon)
```

# Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
beta	beta. Default value = 8.0
ro	ro. Default value = $0.001$
epsilon	epsilon. Default value = $0.001$

# Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::PW_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

QAR\_CIP\_NSGAII\_A

QAR\_CIP\_NSGAII\_A KEEL Association Rules Algorithm

#### **Description**

QAR\_CIP\_NSGAII\_A Association Rules Algorithm from KEEL.

## Usage

```
QAR_CIP_NSGAII_A(dat, seed, NumberofObjetives, NumberofEvaluations, PopulationSize, ProbabilityofMutation, Thefactorofamplitudeforeachattributeofthedataset, Differencethreshold)
```

#### **Arguments**

dat Dataset as a data.frame object

seed. Default value = 1286082570

NumberofObjetives

Number of Objetives. Default value = 3

NumberofEvaluations

Number of Evaluations. Default value = 50000

PopulationSize Population Size. Default value = 100

ProbabilityofMutation

Probability of Mutation. Default value = 0.1

The factor of amplitude for each attribute of the dataset

The factor of amplitude for each attribute of the dataset. Default value = 2.0

Differencethreshold

Difference threshold. Default value = 5.0

#### **Details**

\$run() Run algorith

\$showRules(numRules) Show a number of rules. By default all rules.

\$getInterestMeasures() Return a data.frame with all interest measures of set rules.

\$sortBy(interestMeasure) Order set rules by interest measure.

\$writeCSV(fileName, sep) Create CSV file with set rules. Default fileName="rules" sep=","

\$writePMML(fileName) Create PMML file with set rules. Default fileName="rules"

\$addInterestMeasure(name, colName) Add interest measures to set rules. Some interest measures supported:

QAR\_CIP\_NSGAII\_A 151

```
"allConfidence" (Omiencinski, 2003)
"crossSupportRatio", cross-support ratio (Xiong et al., 2003)
"lift", interest factor (Brin et al. 1997)
"support", supp (Agrawal et al., 1996)
"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002)
"chiSquared", X^2 (Liu et al., 1999)
"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002)
"collectiveStrength"
"confidence", conf (Agrawal et al., 1996)
"conviction" (Brin et al. 1997)
"cosine" (Tan et al., 2004)
"coverage", cover, LHS-support
"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)
"casualConfidence", casual confidence (Kodratoff, 1999)
"casualSupport", casual support (Kodratoff, 1999)
"counterexample", example and counterexample rate
"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)
"doc", difference of confidence (Hofmann and Wilhelm, 2001)
"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)
"gini", Gini index (Tan et al., 2004)
"hyperLift" (Hahsler and Hornik, 2007)
"hyperConfidence" (Hahsler and Hornik, 2007)
"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)
```

```
"implicationIndex", implication index (Gras, 1996)
"improvement" (Bayardo et al., 2000)
"jaccard", Jaccard coefficient (Tan and Kumar, 2000)
"iMeasure", J-measure, J (Smyth and Goodman, 1991)
"kappa" (Tan and Kumar, 2000)
"klosgen", Klosgen (Tan and Kumar, 2000)
"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)
"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)
"laplace", L (Tan and Kumar 2000)
"leastContradiction", least contradiction (Aze and Kodratoff, 2004
"lerman", Lerman similarity (Lerman, 1981)
"leverage", PS (Piatetsky-Shapiro 1991)
"mutualInformation", uncertainty, M (Tan et al., 2002)
"oddsRatio", odds ratio alpha (Tan et al., 2004)
"phi", correlation coefficient phi (Tan et al. 2004)
"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)
"RLD", relative linkage disequilibrium (Kenett and Salini, 2008)
"sebag", Sebag measure (Sebag and Schoenauer, 1988)
"support", supp (Agrawal et al., 1996)
"varyingLiaison", varying rates liaison (Bernard and Charron, 1996)
"yuleQ", Yule's Q (Tan and Kumar, 2000)
"yuleY", Yule's Y (Tan and Kumar, 2000)
For more information see ?arules::interestMeasure
```

QDA\_C 153

## Value

A arules class with the Association Rules for both dat dataset.

#### **Examples**

```
#Load KEEL dataset
dat<-RKEEL::loadKeelDataset("car")</pre>
#Create algorithm
algorithm <- RKEEL::QAR_CIP_NSGAII_A(dat)</pre>
#Run algorithm
algorithm$run()
#Rules in format arules
algorithm$rules
#Show a number of rules
algorithm$showRules(2)
#Return a data.frame with all interest measures of set rules
algorithm$getInterestMeasures()
#Add interst measure YuleY to set rules
algorithm$addInterestMeasure("YuleY", "yulesY")
#Sort by interest measure lift
algorithm$sortBy("lift")
#Save rules in CSV file
algorithm$writeCSV(paste0(tempdir(), "/myrules"))
```

QDA\_C

QDA\_C KEEL Classification Algorithm

# Description

QDA\_C Classification Algorithm from KEEL.

# Usage

```
QDA_C(train, test, seed)
```

# Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

154 RBFN\_C

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::QDA_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

RBFN\_C

RBFN\_C KEEL Classification Algorithm

# Description

RBFN\_C Classification Algorithm from KEEL.

# Usage

```
RBFN_C(train, test, neurons, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object

neurons. Default value = 50

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

RBFN\_R 155

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::RBFN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

RBFN\_R

RBFN\_R KEEL Regression Algorithm

## **Description**

RBFN\_R Regression Algorithm from KEEL.

### Usage

```
RBFN_R(train, test, neurons, seed)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
neurons Default value = 50

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

## Value

A data.frame with the actual and predicted values for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::RBFN_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

156 RegressionResults

read.keel

Read keel dataset

#### **Description**

Method for read datasets in .dat KEEL format

#### Usage

read.keel(file)

## **Arguments**

file

File containing the dataset to be read. It must be in KEEL .dat format.

#### Value

Returns a data.frame object with the dataset

RegressionAlgorithm

Regression Algorithm

## Description

Class inheriting of KeelAlgorithm, to common methods for all KEEL Regression Algorithms. The specific regression algorithms must inherit of this class.

The run() method receives three parameters. The folderPath parameter indicates where to place the folder with the experiments if wanted. If it is not indicated, the folder is placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

RegressionResults

Regression Results

#### **Description**

Class to calculate and store some results for a RegressionAlgorithm. It receives as parameter the prediction of a regression algorithm as a data.frame object.

Relief\_FS 157

Relief\_FS

Relief\_FS KEEL Preprocess Algorithm

## **Description**

Relief\_FS Preprocess Algorithm from KEEL.

#### Usage

```
Relief_FS(train, test, paramKNN, relevanceThreshold,
    numInstancesSampled, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
paramKNN paramKNN. Default value = 1

relevanceThreshold

relevanceThreshold. Default value = 0.20

 ${\tt numInstancesSampled}$ 

numInstancesSampled. Default value = 1000

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::Relief_FS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

Ripper\_C

D:	<b>.</b>			^
Ri	μ	υe	Γ_	Ŀ

Ripper\_C KEEL Classification Algorithm

# Description

Ripper\_C Classification Algorithm from KEEL.

## Usage

```
Ripper_C(train, test, grow_pct, k, seed)
```

#### **Arguments**

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Ripper_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

RISE\_C 159

RISE\_C

RISE\_C KEEL Classification Algorithm

## **Description**

RISE\_C Classification Algorithm from KEEL.

## Usage

```
RISE_C(train, test, Q, S)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
Q Q. Default value = 1
S S. Default value = 2

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::RISE_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

runCV

Run Cross-Validation

# Description

Run a cross-validation experiment

## Usage

```
runCV(algorithm, dataset, numFolds, cores)
```

160 runParallel

# Arguments

algorithm Algorithm to be executed in the CV. It must has the parameters to be used in the

executions.

dataset Dataset to perform the CV. It is divided in numFolds disjoint partitions and in

each iteration, one is used for test and the rest for train.

numFolds Number of folds for the cross-validation procedure.

cores Number of cores to execute in parallel. If it is missed, default value is 1 (se-

quential execution).

#### Value

Returns a list with the mean results of the numFolds executions.

## **Examples**

```
#Load datasets
iris <- RKEEL::loadKeelDataset("iris")

#Create algorithm
learner_C45_C <- RKEEL::C45_C(iris, iris)

#Perform 5-folds CV
results <- RKEEL::runCV(learner_C45_C, iris, 5)</pre>
```

runParallel

Run Parallel

## **Description**

Run a set of RKEEL algorithms in parallel

#### Usage

```
runParallel(algorithmList, cores)
```

# Arguments

algorithmList List of RKEEL Algorithms to be executed

cores Number of cores to execute in parallel. If it is not specified, it detects the cores

automatically and execute the experiment in all of them

#### Value

Returns a list with the executed algorithms

runSequential 161

#### **Examples**

```
#Load datasets
iris_train <- RKEEL::loadKeelDataset("iris_train")
iris_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithms
learner_C45_C <- RKEEL::C45_C(iris_train, iris_test)
learner_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)

#Create list
algorithms <- list(learner_C45_C, learner_KNN_C, learner_Logistic_C, learner_LDA_C)

#Run algorithms in parallel in two cores
par <- RKEEL::runParallel(algorithms, 2)</pre>
```

runSequential

Run Sequential

#### **Description**

Run a set of RKEEL algorithms in sequential.

## Usage

```
runSequential(algorithmList)
```

## **Arguments**

algorithmList List of RKEEL Algorithms to be executed

#### Value

Returns a list with the executed algorithms

```
#Load datasets
iris_train <- RKEEL::loadKeelDataset("iris_train")
iris_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithms
learner_C45_C <- RKEEL::C45_C(iris_train, iris_test)
learner_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)</pre>
```

162 SaturationFilter\_F

```
#Create list
algorithms <- list(learner_C45_C, learner_KNN_C, learner_Logistic_C,
    learner_LDA_C)
#Run algorithms
seq <- RKEEL::runSequential(algorithms)</pre>
```

SaturationFilter\_F

SaturationFilter\_F KEEL Preprocess Algorithm

## **Description**

SaturationFilter\_F Preprocess Algorithm from KEEL.

## Usage

```
SaturationFilter_F(train, test, seed)
```

# **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::SaturationFilter_F(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

SFS\_IEP\_FS 163

SFS\_IEP\_FS

SFS\_IEP\_FS KEEL Preprocess Algorithm

## **Description**

SFS\_IEP\_FS Preprocess Algorithm from KEEL.

## Usage

```
SFS_IEP_FS(train, test, threshold, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
threshold threshold. Default value = 0.005

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::SFS_IEP_FS(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

SGA\_C

SGA\_C KEEL Classification Algorithm

## Description

SGA\_C Classification Algorithm from KEEL.

164 SGA\_C

#### Usage

```
SGA_C(train, test, mut_prob_1to0, mut_prob_0to1, cross_prob,
   pop_size, evaluations, alfa, selection_type, k,
   distance, seed)
```

#### Arguments

train Train dataset as a data.frame object

test Test dataset as a data.frame object

mut\_prob\_1to0 mut\_prob\_1to0. Default value = 0.01

mut\_prob\_0to1 mut\_prob\_0to1. Default value = 0.001

cross\_prob cross\_prob. Default value = 1

pop\_size pop\_size. Default value = 50

evaluations evaluations. Default value = 10000

alfa alfa. Default value = 0.5

selection\_type selection\_type. Default value = "orden\_based"

k. Default value = 1

distance distance. Default value = "Euclidean"

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

## Value

A data.frame with the actual and predicted classes for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::SGA_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Shrink\_C 165

Shrink\_C

Shrink\_C KEEL Classification Algorithm

#### **Description**

Shrink\_C Classification Algorithm from KEEL.

## Usage

```
Shrink_C(train, test, k, distance)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k. Default value = 1

distance distance. Default value = "Euclidean"

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Shrink_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Slipper\_C

Slipper\_C KEEL Classification Algorithm

## **Description**

Slipper\_C Classification Algorithm from KEEL.

#### Usage

```
Slipper_C(train, test, grow_pct, numBoosting, seed)
```

166 SMO\_C

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
grow\_pct grow\_pct. Default value = 0.66
numBoosting numBoosting. Default value = 100

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

## **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Slipper_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

SMO\_C

SMO\_C KEEL Classification Algorithm

## **Description**

SMO\_C Classification Algorithm from KEEL.

## Usage

```
SMO_C(train, test, C, toleranceParameter, epsilon,
   RBFKernel_gamma, normalized_PolyKernel_exponent,
   normalized_PolyKernel_useLowerOrder, PukKernel_omega,
   PukKernel_sigma, StringKernel_lambda,
   StringKernel_subsequenceLength,
   StringKernel_maxSubsequenceLength, StringKernel_normalize,
   StringKernel_pruning, KernelType, FitLogisticModels,
   ConvertNominalAttributesToBinary, PreprocessType, seed)
```

SMO\_C 167

## Arguments

train Train dataset as a data.frame object

test Test dataset as a data.frame object

C. Default value = 1.0

toleranceParameter

toleranceParameter. Default value = 0.001

epsilon epsilon. Default value = 1.0e-12

RBFKernel\_gamma

RBFKernel\_gamma. Default value = 0.01

normalized\_PolyKernel\_exponent

normalized\_PolyKernel\_exponent. Default value = 1

normalized\_PolyKernel\_useLowerOrder

normalized\_PolyKernel\_useLowerOrder. Default value = FALSE

PukKernel\_omega

PukKernel\_omega. Default value = 1.0

PukKernel\_sigma

PukKernel sigma. Default value = 1.0

StringKernel\_lambda

StringKernel\_lambda. Default value = 0.5

StringKernel\_subsequenceLength

StringKernel\_subsequenceLength. Default value = 3

StringKernel\_maxSubsequenceLength

StringKernel\_maxSubsequenceLength. Default value = 9

StringKernel\_normalize

StringKernel\_normalize. Default value = FALSE

StringKernel\_pruning

StringKernel\_pruning. Default value = "None"

KernelType KernelType. Default value = "PolyKernel"

FitLogisticModels

FitLogisticModels. Default value = FALSE

ConvertNominalAttributesToBinary

ConvertNominalAttributesToBinary. Default value = TRUE

PreprocessType PreprocessType. Default value = "Normalize"

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::SMO_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

# Description

SSGA\_Integer\_knn\_FS Preprocess Algorithm from KEEL.

## Usage

```
SSGA_Integer_knn_FS(train, test, paramKNN, nEval, pop_size,
   numFeatures, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object paramKNN paramKNN. Default value = 1 nEval nEval. Default value = 5000 pop\_size pop\_size. Default value = 100 numFeatures numFeatures. Default value = 3

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

 $Tan\_GP\_C$ 

### **Examples**

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::SSGA_Integer_knn_FS(data_train, data_test)
algorithm <- RKEEL::SSGA_Integer_knn_FS(data_train, data_test, nEval = 10, pop_size = 10)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

Tan\_GP\_C

Tan\_GP\_C KEEL Classification Algorithm

#### **Description**

Tan\_GP\_C Classification Algorithm from KEEL.

## Usage

```
Tan_GP_C(train, test, population_size, max_generations,
   max_deriv_size, rec_prob, mut_prob, copy_prob, w1, w2,
   elitist_prob, support, seed)
```

## **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object population\_size

population\_size. Default value = 150

max\_generations

 $max\_generations$ . Default value = 100

max\_deriv\_size max\_deriv\_size. Default value = 20

rec\_prob rec\_prob. Default value = 0.8
mut\_prob mut\_prob. Default value = 0.1
copy\_prob copy\_prob. Default value = 0.01

w1 w1. Default value = 0.7w2 w2. Default value = 0.8

elitist\_prob elitist\_prob. Default value = 0.06 support support. Default value = 0.03

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

Thrift\_R

#### Value

A data frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::Tan_GP_C(data_train, data_test)
algorithm <- RKEEL::Tan_GP_C(data_train, data_test, population_size = 5, max_generations = 10)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

Thrift\_R

Thrift\_R KEEL Regression Algorithm

#### **Description**

Thrift\_R Regression Algorithm from KEEL.

#### Usage

```
Thrift_R(train, test, numLabels, popSize, evaluations,
    crossProb, mutProb, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numLabels numLabels. Default value = 3
popSize popSize. Default value = 61

evaluations evaluations. Default value = 10000 crossProb crossProb. Default value = 0.6 mutProb mutProb. Default value = 0.1

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

UniformFrequency\_D 171

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::Thrift_R(data_train, data_test)
algorithm <- RKEEL::Thrift_R(data_train, data_test, popSize = 5, evaluations = 10)

#Run algorithm
algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

UniformFrequency\_D

UniformFrequency\_D KEEL Preprocess Algorithm

# Description

UniformFrequency\_D Preprocess Algorithm from KEEL.

#### Usage

```
UniformFrequency_D(train, test, numIntervals, seed)
```

## **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numIntervals numIntervals. Default value = 10

seed Seed for random numbers. If it is not assigned a value, the seed will be a random

number

#### Value

A data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::UniformFrequency_D(data_train, data_test)

#Run algorithm
algorithm$run()</pre>
```

UniformWidth\_D

```
#See results
algorithm$preprocessed_test
```

UniformWidth\_D

UniformWidth\_D KEEL Preprocess Algorithm

## **Description**

UniformWidth\_D Preprocess Algorithm from KEEL.

## Usage

```
UniformWidth_D(train, test, numIntervals)
```

# Arguments

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numIntervals numIntervals. Default value = 10

### Value

 $\boldsymbol{A}$  data.frame with the preprocessed data for both train and test datasets.

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::UniformWidth_D(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

VWFuzzyKNN\_C 173

VWFuzzyKNN\_C

VWFuzzyKNN\_C KEEL Classification Algorithm

#### **Description**

VWFuzzyKNN\_C Classification Algorithm from KEEL.

## Usage

```
VWFuzzyKNN_C(train, test, k, init_k)
```

#### **Arguments**

train Train dataset as a data.frame object test Test dataset as a data.frame object

k k. Default value = 3
init\_k init\_k. Default value = 3

#### Value

A data.frame with the actual and predicted classes for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("iris_train")
data_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithm
algorithm <- RKEEL::VWFuzzyKNN_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $WM_R$ 

WM\_R KEEL Regression Algorithm

## **Description**

WM\_R Regression Algorithm from KEEL.

#### Usage

```
WM_R(train, test, numlabels, KB)
```

174 writeDatFromDataframe

### **Arguments**

train Train dataset as a data.frame object
test Test dataset as a data.frame object
numlabels numlabels. Default value = 5
KB KB. Default value = FALSE

#### Value

A data.frame with the actual and predicted values for both train and test datasets.

#### **Examples**

```
data_train <- RKEEL::loadKeelDataset("autoMPG6_train")
data_test <- RKEEL::loadKeelDataset("autoMPG6_test")

#Create algorithm
algorithm <- RKEEL::WM_R(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions</pre>
```

 $write {\tt DatFromDataframe} \ \ \textit{Write .dat from data.frame}$ 

# Description

Method for writing a .dat dataset file in KEEL format given a data.frame dataset

## Usage

```
writeDatFromDataframe(data, fileName)
```

## **Arguments**

data data.frame dataset

fileName String with the file name to store the dataset

```
data(iris)
writeDatFromDataframe(iris, paste0(tempdir(), "/iris.dat"))
```

writeDatFromDataframes 175

writeDatFromDataframes

Write .dat from data.frames

# Description

Method for writing both train and test .dat dataset files in KEEL format.

## Usage

```
writeDatFromDataframes(trainData, testData,
    trainFileName, testFileName)
```

## **Arguments**

trainData Train data as data.frame object testData Test data as data.frame object

trainFileName String with the file name to store the train dataset testFileName String with the file name to store the test dataset

ZScore\_TR KEEL Preprocess Algorithm

# Description

ZScore\_TR Preprocess Algorithm from KEEL.

### Usage

```
ZScore_TR(train, test)
```

# Arguments

train Train dataset as a data.frame object test Test dataset as a data.frame object

#### Value

A data.frame with the preprocessed data for both train and test datasets.

ZScore\_TR

```
data_train <- RKEEL::loadKeelDataset("car_train")
data_test <- RKEEL::loadKeelDataset("car_test")

#Create algorithm
algorithm <- RKEEL::ZScore_TR(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test</pre>
```

# **Index**

* association rules	ClassificationResults, 35
Alatasetal_A, 8	CMAR_C, 37
Alcalaetal_A, 12	CNN_C, 38
Apriori_A, 18	CPAR_C, 39
AssociationRulesAlgorithm, 22	CPW_C, 40
EARMGA_A, 48	CW_C, 41
Eclat_A, 51	DecrRBFN_C, 43
FPgrowth_A, 58	Deeps_C, 44
FuzzyApriori_A,64	DSM_C, 46
GAR_A, 72	DT_GA_C, 47
GENAR_A, 76	Falco_GP_C, 56
GeneticFuzzyApriori_A,83	FCRA_C, 57
GeneticFuzzyAprioriDC_A, 79	FRNN_C, 61
MODENAR_A, 117	FURIA_C, 63
MOEA_Ghosh_A, 121	FuzzyFARCHD_C, 67
MOPNAR_A, 124	FuzzyKNN_C, 68
NICGAR_A, 130	FuzzyNPC_C, 69
QAR_CIP_NSGAII_A, 150	GANN_C, 70
* classification	GFS_AdaBoost_C, 89
AdaBoost_I, 7	GFS_LogitBoost_C, 93
AdaBoostNC_C, 6	ID3_C, 96
ART_C, 21	IF_KNN_C, 97
AssociativeClassificationAlgorithm,	${\tt ImbalancedClassificationAlgorithm},$
22	99
BNGE_C, 23	IncrRBFN_C, 99
Bojarczuk_GP_C, 24	JFKNN_C, 101
BSE_C, 25	Kernel_C, 102
C45_C, 28	KNN_C, 104
C45Binarization_C, 26	KSNN_C, 105
C45Rules_C, 27	KStar_C, 106
C_SVM_C, 42	LDA_C, 107
CamNN_C, 29	LinearLMS_C, 108
CART_C, 29	Logistic_C, 110
CBA_C, 31	MLP_BP_C, 114
CenterNN_C, 32	NB_C, 129
CFAR_C, 32	NM_C, 133
CFKNN_C, 33	NNEP_C, 134
CHC_C, 34	NU_SVM_C, 136
ClassificationAlgorithm, 35	PART_C, 138

PDFC_C, 138	* regression
PFKNN_C, 140	CART_R, 30
PNN_C, 140	EPSILON_SVR_R, 55
PolQuadraticLMS_C, 141	FRSBM_R, 62
PRISM_C, 144	GFS_GP_R, 90
PSO_ACO_C, 146	GFS_GSP_R, 91
PUBLIC_C, 148	GFS_RB_MF_R, 94
PW_C, 149	LinearLMS_R, 108
QDA_C, 153	M5_R, 112
RBFN_C, 154	M5Rules_R, 111
Ripper_C, 158	MLP_BP_R, 115
RISE_C, 159	NU_SVR_R, 137
SGA_C, 163	PolQuadraticLMS_R, 142
Shrink_C, 165	RBFN_R, 155
Slipper_C, 165	RegressionAlgorithm, 156
SMO_C, 166	RegressionResults, 156
Tan_GP_C, 169	Thrift_R, 170
VWFuzzyKNN_C, 173	WM_R, 173
rm d22yddd_5, 175	* utils
ABB_IEP_FS, 5	downloadFromMirror,45
AllKNN_TSS, 15	<pre>getAttributeLinesFromDataframes,</pre>
AllPosible_MV, 16	87
ANR_F, 17	hasContinuousData,95
Bayesian_D, 23	hasMissingValues, 95
CleanAttributes_TR, 36	isMultiClass, 100
ClusterAnalysis_D, 36	loadKeelDataset, 109
DecimalScaling_TR, 43	read.keel, 156
ID3_D, 96	runCV, 159
Ignore_MV, 98	runParallel, 160
IterativePartitioningFilter_F, 100	runSequential, 161
KMeans_MV, 103	writeDatFromDataframe, 174
KNN_MV, 104	writeDatFromDataframes, 175
LVF_IEP_FS, 110	
MinMax_TR, 113	ABB_IEP_FS, 5
ModelCS_TSS, 116	AdaBoost_I, 7
MostCommon_MV, 128	AdaBoostNC_C, 6
	Alatasetal_A, 8
Nominal2Binary_TR, 135 POP_TSS, 143	Alcalaetal_A, 12
PreprocessAlgorithm, 144	AllKNN_TSS, 15
Proportional_D, 145	AllPosible_MV, 16
PSRCG_TSS, 147	ANR_F, 17
	Apriori_A, 18
Relief_FS, 157	ART_C, 21
SaturationFilter_F, 162	AssociationRulesAlgorithm, 22
SFS_IEP_FS, 163	AssociativeClassificationAlgorithm, 22
SSGA_Integer_knn_FS, 168	Davissian D 22
UniformFrequency_D, 171	Bayesian_D, 23
UniformWidth_D, 172	BNGE_C, 23
ZScore_TR, 175	Bojarczuk_GP_C, 24

BSE_C, 25	GeneticFuzzyApriori_A,83
	GeneticFuzzyAprioriDC_A, 79
C45_C, 28	<pre>getAttributeLinesFromDataframes, 87</pre>
C45Binarization_C, 26	getExePath, 88
C45Rules_C, 27	getJarList,88
C_SVM_C, 42	getJarPath, 89
CamNN_C, 29	GFS_AdaBoost_C, 89
CART_C, 29	GFS_GP_R, 90
CART_R, 30	GFS_GSP_R, 91
CBA_C, 31	GFS_LogitBoost_C, 93
CenterNN_C, 32	GFS_RB_MF_R, 94
CFAR_C, 32	
CFKNN_C, 33	hasContinuousData, 95
CHC_C, 34	hasMissingValues, 95
ClassificationAlgorithm, 35	
ClassificationResults, 35	ID3_C, 96
CleanAttributes_TR, 36	ID3_D, 96
ClusterAnalysis_D, 36	IF_KNN_C, 97
CMAR_C, 37	Ignore_MV,98
CNN_C, 38	ImbalancedClassificationAlgorithm, 99
CPAR_C, 39	IncrRBFN_C, 99
CPW_C, 40	isMultiClass, 100
CW_C, 40	<pre>IterativePartitioningFilter_F, 100</pre>
CW_C, 41	
DecimalScaling_TR, 43	JFKNN_C, 101
DecrRBFN_C, 43	
Deeps_C, 44	KeelAlgorithm, 102
downloadFromMirror, 45	Kernel_C, 102
DSM_C, 46	KMeans_MV, 103
	KNN_C, 104
DT_GA_C, 47	KNN_MV, 104
EARMGA_A, 48	KSNN_C, 105
Eclat_A, 51	KStar_C, 106
	LDA C 107
EPSILON_SVR_R, 55	LDA_C, 107
Falco_GP_C, 56	LinearLMS_C, 108
FCRA_C, 57	LinearLMS_R, 108
	loadKeelDataset, 109
FPgrowth_A, 58	Logistic_C, 110
FRNN_C, 61	LVF_IEP_FS, 110
FRSBM_R, 62	ME D 112
FURIA_C, 63	M5_R, 112
FuzzyApriori_A, 64	M5Rules_R, 111
FuzzyFARCHD_C, 67	MinMax_TR, 113
FuzzyKNN_C, 68	MLP_BP_C, 114
FuzzyNPC_C, 69	MLP_BP_R, 115
CANIN C 70	ModelCS_TSS, 116
GANN_C, 70	MODENAR_A, 117
GAR_A, 72	MOEA_Ghosh_A, 121
GENAR_A, 76	MOPNAR_A, 124

MostCommon_MV, 128	R6_CART_R (CART_R), 30
	R6_CBA_C (CBA_C), 31
NB_C, 129	R6_CenterNN_C (CenterNN_C), 32
NICGAR_A, 130	R6_CFAR_C (CFAR_C), 32
NM_C, 133	R6_CFKNN_C (CFKNN_C), 33
NNEP_C, 134	R6_CHC_C (CHC_C), 34
Nominal2Binary_TR, 135	R6_CleanAttributes_TR
NU_SVM_C, 136	(CleanAttributes_TR), 36
NU_SVR_R, 137	R6_ClusterAnalysis_D
	(ClusterAnalysis_D), 36
PART_C, 138	R6_CMAR_C (CMAR_C), 37
PDFC_C, 138	R6_CNN_C (CNN_C), 38
PFKNN_C, 140	R6_CPAR_C (CPAR_C), 39
PNN_C, 140	R6_CPW_C (CPW_C), 40
PolQuadraticLMS_C, 141	R6_CW_C (CW_C), 41
PolQuadraticLMS_R, 142	R6_DecimalScaling_TR
POP_TSS, 143	(DecimalScaling_TR), 43
PreprocessAlgorithm, 144	R6_DecrRBFN_C (DecrRBFN_C), 43
PRISM_C, 144	R6_Deeps_C (Deeps_C), 44
Proportional_D, 145	R6_DSM_C (DSM_C), 46
PSO_ACO_C, 146	R6_DT_GA_C (DT_GA_C), 47
PSRCG_TSS, 147	R6_EARMGA_A (EARMGA_A), 48
PUBLIC_C, 148	R6_Eclat_A (Eclat_A), 51
PW_C, 149	R6_EPSILON_SVR_R (EPSILON_SVR_R), 55
	R6_Falco_GP_C (Falco_GP_C), 56
QAR_CIP_NSGAII_A, 150	R6_FCRA_C (FCRA_C), 57
QDA_C, 153	R6_FPgrowth_A (FPgrowth_A), 58
	R6_FRNN_C (FRNN_C), 61
R6_ABB_IEP_FS (ABB_IEP_FS), 5	R6_FRSBM_R (FRSBM_R), 62
R6_AdaBoost_I (AdaBoost_I), 7	R6_FURIA_C (FURIA_C), 63
R6_AdaBoostNC_C (AdaBoostNC_C), 6	
R6_Alatasetal_A (Alatasetal_A), 8	R6_FuzzyApriori_A (FuzzyApriori_A), 64
R6_Alcalaetal_A (Alcalaetal_A), 12	R6_FuzzyFARCHD_C (FuzzyFARCHD_C), 67
R6_AllKNN_TSS (AllKNN_TSS), 15	R6_FuzzyKNN_C (FuzzyKNN_C), 68
R6_AllPosible_MV (AllPosible_MV), 16	R6_FuzzyNPC_C (FuzzyNPC_C), 69
R6_ANR_F (ANR_F), 17	R6_GANN_C (GANN_C), 70
R6_Apriori_A (Apriori_A), 18	R6_GAR_A (GAR_A), 72
R6_ART_C (ART_C), 21	R6_GENAR_A (GENAR_A), 76
R6_Bayesian_D (Bayesian_D), 23	R6_GeneticFuzzyApriori_A
R6_BNGE_C (BNGE_C), 23	(GeneticFuzzyApriori_A), 83
R6_Bojarczuk_GP_C (Bojarczuk_GP_C), 24	R6_GeneticFuzzyAprioriDC_A
R6_BSE_C (BSE_C), 25	(GeneticFuzzyAprioriDC_A), 79
R6_C45_C (C45_C), 28	R6_GFS_AdaBoost_C (GFS_AdaBoost_C), 89
R6_C45Binarization_C	R6_GFS_GP_R (GFS_GP_R), 90
$(C45Binarization_C), 26$	R6_GFS_GSP_R (GFS_GSP_R), 91
R6_C45Rules_C (C45Rules_C), 27	R6_GFS_LogitBoost_C (GFS_LogitBoost_C)
$R6_C_SVM_C(C_SVM_C)$ , 42	93
R6_CamNN_C (CamNN_C), 29	R6_GFS_RB_MF_R (GFS_RB_MF_R), 94
R6_CART_C (CART_C), 29	R6_ID3_C (ID3_C), 96

R6_ID3_D (ID3_D), 96	R6_PS0_ACO_C (PS0_ACO_C), 146
R6_IF_KNN_C (IF_KNN_C), 97	R6_PSRCG_TSS (PSRCG_TSS), 147
R6_Ignore_MV (Ignore_MV), 98	R6_PUBLIC_C (PUBLIC_C), 148
R6_IncrRBFN_C (IncrRBFN_C), 99	R6_PW_C (PW_C), 149
R6_IterativePartitioningFilter_F	R6_QAR_CIP_NSGAII_A (QAR_CIP_NSGAII_A),
<pre>(IterativePartitioningFilter_F),</pre>	150
100	R6_QDA_C (QDA_C), 153
R6_JFKNN_C (JFKNN_C), 101	R6_RBFN_C (RBFN_C), 154
R6_Kernel_C (Kernel_C), 102	R6_RBFN_R (RBFN_R), 155
R6_KMeans_MV (KMeans_MV), 103	R6_Relief_FS (Relief_FS), 157
R6_KNN_C (KNN_C), 104	R6_Ripper_C (Ripper_C), 158
R6_KNN_MV (KNN_MV), 104	R6_RISE_C (RISE_C), 159
R6_KSNN_C (KSNN_C), 105	R6_SaturationFilter_F
R6_KStar_C (KStar_C), 106	(SaturationFilter_F), 162
R6_LDA_C (LDA_C), 107	R6_SFS_IEP_FS (SFS_IEP_FS), 163
R6_LinearLMS_C (LinearLMS_C), 108	R6_SGA_C (SGA_C), 163
R6_LinearLMS_R (LinearLMS_R), 108	R6_Shrink_C (Shrink_C), 165
R6_Logistic_C (Logistic_C), 110	R6_Slipper_C (Slipper_C), 165
R6_LVF_IEP_FS (LVF_IEP_FS), 110	R6_SMO_C (SMO_C), 166
R6_M5_R (M5_R), 112	R6_SSGA_Integer_knn_FS
R6_M5Rules_R (M5Rules_R), 111	(SSGA_Integer_knn_FS), 168
R6_MinMax_TR (MinMax_TR), 113	R6_Tan_GP_C (Tan_GP_C), 169
R6_MLP_BP_C (MLP_BP_C), 114	R6_Thrift_R (Thrift_R), 170
R6_MLP_BP_R (MLP_BP_R), 115	R6_UniformFrequency_D
R6_ModelCS_TSS (ModelCS_TSS), 116	(UniformFrequency_D), 171
R6_MODENAR_A (MODENAR_A), 117	R6_UniformWidth_D (UniformWidth_D), 172
R6_MOEA_Ghosh_A (MOEA_Ghosh_A), 121	R6_VWFuzzyKNN_C (VWFuzzyKNN_C), 173
R6_MOPNAR_A (MOPNAR_A), 124	R6_WM_R (WM_R), 173
R6_MostCommon_MV (MostCommon_MV), 128	R6_ZScore_TR (ZScore_TR), 175
R6_NB_C (NB_C), 129	RBFN_C, 154
R6_NICGAR_A (NICGAR_A), 130	RBFN_R, 155
R6_NM_C (NM_C), 133	read.keel, 156
R6_NNEP_C (NNEP_C), 134	RegressionAlgorithm, 156
R6_Nominal2Binary_TR	RegressionResults, 156
(Nominal2Binary_TR), 135	Relief_FS, 157
R6_NU_SVM_C (NU_SVM_C), 136	Ripper_C, 158
R6_NU_SVR_R (NU_SVR_R), 137	RISE_C, 159
R6_PART_C (PART_C), 138	runCV, 159
R6_PDFC_C (PDFC_C), 138	runParallel, 160
R6_PFKNN_C (PFKNN_C), 140	runSequential, 161
R6_PNN_C (PNN_C), 140	
R6_PolQuadraticLMS_C	SaturationFilter_F, 162
(PolQuadraticLMS_C), 141	SFS_IEP_FS, 163
R6_PolQuadraticLMS_R	SGA_C, 163
(PolQuadraticLMS_R), 142	Shrink_C, 165
R6_POP_TSS (POP_TSS), 143	Slipper_C, 165
R6_PRISM_C (PRISM_C), 144	SMO_C, 166
R6_Proportional_D (Proportional_D), 145	SSGA_Integer_knn_FS, 168

```
Tan_GP_C, 169
Thrift_R, 170
UniformFrequency_D, 171
UniformWidth_D, 172
VWFuzzyKNN_C, 173
WM_R, 173
writeDatFromDataframe, 174
writeDatFromDataframes, 175
ZScore_TR, 175
```