

# Package ‘RKEEL’

May 13, 2021

**Type** Package

**Title** Using KEEL in R Code

**Version** 1.3.3

**Depends** R (>= 3.4.0)

**Date** 2021-05-13

**Author** Jose M. Moyano [aut, cre],  
Luciano Sanchez [aut],  
Oliver Sanchez [ctb],  
Jesus Alcalá-Fernández [ctb]

**Maintainer** Jose M. Moyano <jmoyano@uco.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 <<http://www.keel.es/>>.

**SystemRequirements** Java (>= 8)

**License** GPL

**Imports** R6, XML, doParallel, foreach, gdata, RKEELjars (>= 1.0.19),  
RKEELdata (>= 1.0.5), pmml, arules, Matrix, rJava

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2021-05-13 14:12:11 UTC

## R topics documented:

ABB_IEP_FS . . . . .	5
AdaBoostNC_C . . . . .	6

AdaBoost_I . . . . .	7
Alatasetal_A . . . . .	8
Alcalaetal_A . . . . .	12
AllKNN_TSS . . . . .	15
AllPossible_MV . . . . .	16
ANR_F . . . . .	17
Apriori_A . . . . .	18
ART_C . . . . .	21
AssociationRulesAlgorithm . . . . .	22
AssociativeClassificationAlgorithm . . . . .	22
Bayesian_D . . . . .	23
BNGE_C . . . . .	23
Bojarczuk_GP_C . . . . .	24
BSE_C . . . . .	25
C45Binarization_C . . . . .	26
C45Rules_C . . . . .	27
C45_C . . . . .	28
CamNN_C . . . . .	29
CART_C . . . . .	29
CART_R . . . . .	30
CBA_C . . . . .	31
CenterNN_C . . . . .	32
CFAR_C . . . . .	32
CFKNN_C . . . . .	33
CHC_C . . . . .	34
ClassificationAlgorithm . . . . .	35
ClassificationResults . . . . .	35
CleanAttributes_TR . . . . .	36
ClusterAnalysis_D . . . . .	36
CMAR_C . . . . .	37
CNN_C . . . . .	38
CPAR_C . . . . .	39
CPW_C . . . . .	40
CW_C . . . . .	41
C_SVM_C . . . . .	42
DecimalScaling_TR . . . . .	43
DecrRBFN_C . . . . .	43
Deeps_C . . . . .	44
DSM_C . . . . .	45
DT_GA_C . . . . .	46
EARMGA_A . . . . .	47
Eclat_A . . . . .	51
EPSILON_SVR_R . . . . .	54
Falco_GP_C . . . . .	55
FCRA_C . . . . .	56
FPgrowth_A . . . . .	57
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 . . . . .	71
GENAR_A . . . . .	75
GeneticFuzzyAprioriDC_A . . . . .	79
GeneticFuzzyApriori_A . . . . .	83
getAttributeLinesFromDataframes . . . . .	87
GFS_AdaBoost_C . . . . .	87
GFS_GP_R . . . . .	88
GFS_GSP_R . . . . .	89
GFS_LogitBoost_C . . . . .	91
GFS_RB_MF_R . . . . .	92
hasContinuousData . . . . .	93
hasMissingValues . . . . .	93
ID3_C . . . . .	94
ID3_D . . . . .	94
IF_KNN_C . . . . .	95
Ignore_MV . . . . .	96
ImbalancedClassificationAlgorithm . . . . .	97
IncrRBFN_C . . . . .	97
isMultiClass . . . . .	98
IterativePartitioningFilter_F . . . . .	98
JFKNN_C . . . . .	99
KeelAlgorithm . . . . .	100
Kernel_C . . . . .	100
KMeans_MV . . . . .	101
KNN_C . . . . .	102
KNN_MV . . . . .	102
KSNN_C . . . . .	103
KStar_C . . . . .	104
LDA_C . . . . .	105
LinearLMS_C . . . . .	106
LinearLMS_R . . . . .	106
loadKeelDataset . . . . .	107
Logistic_C . . . . .	108
LVF_IEP_FS . . . . .	108
M5Rules_R . . . . .	109
M5_R . . . . .	110
MinMax_TR . . . . .	111
MLP_BP_C . . . . .	112
MLP_BP_R . . . . .	113
ModelCS_TSS . . . . .	114
MODENAR_A . . . . .	115
MOEA_Ghosh_A . . . . .	119

MOPNAR_A	122
MostCommon_MV	126
NB_C	127
NICGAR_A	128
NM_C	131
NNEP_C	132
Nominal2Binary_TR	133
NU_SVM_C	134
NU_SVR_R	135
PART_C	136
PDFC_C	136
PFKNN_C	138
PNN_C	138
PolQuadraticLMS_C	139
PolQuadraticLMS_R	140
POP_TSS	141
PreprocessAlgorithm	142
PRISM_C	142
Proportional_D	143
PSO_ACO_C	144
PSRCG_TSS	145
PUBLIC_C	146
PW_C	147
QAR_CIP_NSGAII_A	148
QDA_C	151
RBFN_C	152
RBFN_R	153
read.keel	154
RegressionAlgorithm	154
RegressionResults	154
Relief_FS	155
Ripper_C	156
RISE_C	157
runCV	157
runParallel	158
runSequential	159
SaturationFilter_F	160
SFS_IEP_FS	161
SGA_C	161
Shrink_C	163
Slipper_C	163
SMO_C	164
SSGA_Integer_knn_FS	166
Tan_GP_C	167
Thrift_R	168
UniformFrequency_D	169
UniformWidth_D	170
VWFuzzyKNN_C	171

WM\_R . . . . . 171  
 writeDatFromDataframe . . . . . 172  
 writeDatFromDataframes . . . . . 173  
 ZScore\_TR . . . . . 173

**Index** **175**

ABB\_IEP\_FS *ABB\_IEP\_FS KEEL Preprocess Algorithm*

**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.

**Examples**

```
#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
```

---

AdaBoostNC\_C

*AdaBoostNC\_C KEEL Classification Algorithm*

---

### 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	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.

### Examples

```
#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
```

**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.

**Examples**

```
#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
```

---

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	NumberofEvaluations. 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
ImportanceofRulesSupport	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
ImportanceofIntervalsAmplitude	Importance of Intervals Amplitude. Default value = 0.02
ImportanceofNumberofRecordsAlreadyCovered	Importance of Number of Records Already Covered. Default value = 0.01
AmplitudeFactor	Amplitude Factor. Default value = 2.0



**Details**

`$run()` Run algorithm

`$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)

"kloggen", Kloggen (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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-RKEEL::loadKeelDataset("iris")

#Create algorithm
#algorithm <- RKEEL::Alatasetal_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

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
DecreasingFactorofLthresholdNOTUSED	Decreasing Factor of Lthreshold NOT USED. Default value = 0.1
FactorforParentCentricBLXCrossover	Factor for Parent Centric BLXCrossover. Default value = 1.0
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 algorithm

`$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)

"kloggen", Kloggen (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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- Alcalael_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

AllKNN\_TSS

*AllKNN\_TSS KEEL Preprocess Algorithm*

---

## Description

AllKNN\_TSS Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

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

---

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.



**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

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.

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

 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
NumberofPartitionsforNumericAttributes	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 algorithm

`$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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-loadKeelDataset("iris")
```

```
#Create algorithm
#algorithm <- 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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

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.

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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 placed in 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 placed in 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

*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
```

---

BNGE\_C

*BNGE\_C KEEL Classification Algorithm*

---

**Description**

BNGE\_C Classification Algorithm from KEEL.

**Usage**

```
BNGE_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::BNGE_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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



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$run()

#See results
#algorithm$testPredictions
```

---

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	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::BSE_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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"
scoreFunction	scoreFunction. Default value = "WEIGHTED"
bts	bts. Default value = 0.05

**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::C45Binarization_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

**Examples**

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

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

```
#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

### 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::C45_C(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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	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
```

---

CART\_C

*CART\_C KEEL Classification Algorithm*

---

### **Description**

CART\_C Classification Algorithm from KEEL.

### **Usage**

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

**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)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

**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

```

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.

**Examples**

```

#data <- loadKeelDataset("breast")

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions

```

---

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
```

---

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)
```



**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
```

---

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

**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
```

---

 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
```

---

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 placed in 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.

---

CleanAttributes\_TR      *CleanAttributes\_TR 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
```

---

ClusterAnalysis\_D      *ClusterAnalysis\_D KEEL Preprocess Algorithm*

---

**Description**

ClusterAnalysis\_D Preprocess Algorithm from KEEL.

**Usage**

```
ClusterAnalysis_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::ClusterAnalysis_D(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

 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.

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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	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::CNN_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

### **Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

### **Examples**

```
#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
```



---

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.

## Examples

```
#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
```

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	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.

**Examples**

```
#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
```

---

DecimalScaling\_TR      *DecimalScaling\_TR KEEL Preprocess Algorithm*

---

**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
```

---

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)
```

**Arguments**

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

---

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

**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
```

---

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.

**Examples**

```
#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
```

---

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.

**Examples**

```
#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
```

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
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 algorithm

`$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)

"kloggen", Kloggen (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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- 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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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

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  
                          Minimum Confidence. Default value = 0.8

### Details

`$run()` Run algorithm

`$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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-loadKeelDataset("iris")
```

```

#Create algorithm
#algorithm <- 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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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)

```

## 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 = 3

gamma	gamma. Default value = 0.01
coef0	coef0. Default value = 0.0
nu	nu. Default value = 0.5
p	p. Default value = 1.0
shrinking	shrinking. 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 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::EPSILON_SVR_R(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

### 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
```

---

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	WV. Default value = 1.0
crossover_prob	crossover_prob. Default value = 1.0
mut_prob	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
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)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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**

`$run()` Run algorithm  
`$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)

"kloggen", Kloggen (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 iris
dat<-loadKeelDataset("iris")

#Create algorithm
algorithm <- FPgrowth_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 interest measure YuleY to set rules
```

```
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

FRNN\_C

*FRNN\_C KEEL Classification Algorithm*

---

### **Description**

FRNN\_C Classification Algorithm from KEEL.

### **Usage**

```
FRNN_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::FRNN_C(data_train, data_test)

#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

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
numrules	numrules. Default value = 1
sigma	sigma. Default value = 0.0001
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::FRSBM_R(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

### 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
```

---

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
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 algorithm

`$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 iris
```

```

dat<-loadKeelDataset("iris")

#Create algorithm
algorithm <- FuzzyApriori_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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)

```

## 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 depth\_max. Default value = 3  
 K 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
  
```

---

FuzzyKNN\_C

*FuzzyKNN\_C KEEL Classification Algorithm*

---

### Description

FuzzyKNN\_C Classification Algorithm from KEEL.

### 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 = 3
M	M. Default value = 2.0
initialization	initialization. Default value = "CRISP"
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::FuzzyKNN_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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	M. Default value = 2.0

**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
```

---

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	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
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	P_bp. Default value = 0.25
P_param	P_param. Default value = 0.1
P_struct	P_struct. 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.

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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**

<code>dat</code>	Dataset as a data.frame object
<code>seed</code>	seed. Default value = 1286082570
<code>NumberOfItemsets</code>	Number of Itemsets. Default value = 100
<code>TotalNumberOfEvaluations</code>	Total Number of Evaluations. Default value = 50000
<code>PopulationSize</code>	Population Size. Default value = 100
<code>ProbabilityofSelection</code>	Probability of Selection. Default value = 0.25
<code>ProbabilityofCrossover</code>	Probability of Crossover. Default value = 0.7
<code>ProbabilityofMutation</code>	Probability of Mutation. Default value = 0.1
<code>ImportanceofNumberOfRecordsAlreadyCovered</code>	Importance of Number of Records Already Covered. Default value = 0.4
<code>ImportanceofIntervalsAmplitude</code>	Importance of Intervals Amplitude. Default value = 0.7
<code>ImportanceofNumberOfInvolvedAttributes</code>	Importance of Number of Involved Attributes. Default value = 0.5
<code>AmplitudeFactor</code>	Amplitude Factor. Default value = 2.0
<code>MinimumSupport</code>	Minimum Support. Default value = 0.1
<code>MinimumConfidence</code>	Minimum Confidence. Default value = 0.8

**Details**

`$run()` Run algorithm

`$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)

"kloggen", Kloggen (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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-RKEEL::loadKeelDataset("iris")

#Create algorithm
#algorithm <- RKEEL::GAR_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

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	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

`$run()` Run algorithm  
`$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)

"klogen", Klogen (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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- GENAR_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")

```

---

GeneticFuzzyAprioriDC\_A

*GeneticFuzzyAprioriDC\_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 algorithm

`$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)

"kloggen", Kloggen (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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- GeneticFuzzyAprioriDC_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("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

GeneticFuzzyApriori\_A *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
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 algorithm

`$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)

"kloggen", Kloggen (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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- 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()

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

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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

**Examples**

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

attributeLines <- getAttributeLinesFromDataframes(iris_train, iris_test)
```

---

```
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.

**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
```

---

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, probcrosssga, 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	numRules. Default value = 8
popSize	popSize. Default value = 30
numisland	numisland. Default value = 2
steady	steady. Default value = 1
numIter	numIter. Default value = 100
tourSize	tourSize. Default value = 4
mutProb	mutProb. Default value = 0.01
aplMut	aplMut. Default value = 0.1



probMigra	probMigra. Default value = 0.001
probOptimLocal	probOptimLocal. Default value = 0.00
numOptimLocal	numOptimLocal. Default value = 0
idOptimLocal	idOptimLocal. Default value = 0
nichinggap	nichinggap. Default value = 0
maxindniche	maxindniche. Default value = 8
probintraniche	probintraniche. Default value = 0.75
probcrossga	probcrossga. Default value = 0.5
probmutaga	probmutaga. Default value = 0.5
lenchaingap	lenchaingap. Default value = 10
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
```

---

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)
```

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>numLabels</code>	<code>numLabels</code> . Default value = 3
<code>numRules</code>	<code>numRules</code> . Default value = 8
<code>deltafitsap</code>	<code>deltafitsap</code> . Default value = 0.5
<code>p0sap</code>	<code>p0sap</code> . Default value = 0.5
<code>p1sap</code>	<code>p1sap</code> . Default value = 0.5
<code>amplMut</code>	<code>amplMut</code> . Default value = 0.1
<code>nsubsap</code>	<code>nsubsap</code> . Default value = 10
<code>probOptimLocal</code>	<code>probOptimLocal</code> . Default value = 0.00
<code>numOptimLocal</code>	<code>numOptimLocal</code> . Default value = 0
<code>idOptimLocal</code>	<code>idOptimLocal</code> . Default value = 0
<code>probcrossga</code>	<code>probcrossga</code> . Default value = 0.5
<code>probmutaga</code>	<code>probmutaga</code> . Default value = 0.5
<code>lenchaingap</code>	<code>lenchaingap</code> . Default value = 10
<code>maxtreeheight</code>	<code>maxtreeheight</code> . Default value = 8
<code>numItera</code>	<code>numItera</code> . Default value = 10000
<code>seed</code>	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_GSP_R(data_train, data_test)
#algorithm <- RKEEL::GFS_GSP_R(data_train, data_test, numRules=2, numItera=10, maxtreeheight=2)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

## Examples

```
#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
```

---

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	Test dataset as a data.frame object
numLabels	numLabels. Default value = 3
popSize	popSize. Default value = 50
generations	generations. Default value = 100
crossProb	crossProb. Default value = 0.9
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.

### Examples

```
#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$run()

#See results
#algorithm$testPredictions
```

---

hasContinuousData	<i>Has Continuous Data</i>
-------------------	----------------------------

---

**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)
```

---

hasMissingValues	<i>Has Missing Values</i>
------------------	---------------------------

---

**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.

**Examples**

```
iris <- RKEEL::loadKeelDataset("iris")
hasMissingValues(iris)
```

---

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
```

---

ID3\_D

*ID3\_D KEEL Preprocess Algorithm*

---

**Description**

ID3\_D Preprocess Algorithm from KEEL.

**Usage**

```
ID3_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::ID3_D(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

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

**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
```

---

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.

**Examples**

```
#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
```



---

ImbalancedClassificationAlgorithm  
*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 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::IncrRBFN_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

isMultiClass	<i>Is Multi-class</i>
--------------	-----------------------

---

**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)
```

---

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
itemsetsPerLeaf	itemsetsPerLeaf. 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 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
```

---

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.

**Examples**

```
#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()

#See results
#algorithm$testPredictions
```

---

KeelAlgorithm	<i>Keel Algorithm</i>
---------------	-----------------------

---

### Description

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

---

Kernel_C	<i>Kernel_C KEEL Classification Algorithm</i>
----------	---

---

### 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.

### Examples

```
#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
```

---

`KMeans_MV`*KMeans\_MV KEEL Preprocess Algorithm*

---

**Description**

KMeans\_MV Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>k</code>	k. Default value = 10
<code>error</code>	error. Default value = 100
<code>iterations</code>	iterations. Default value = 100
<code>seed</code>	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::KMeans_MV(data_train, data_test)

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test
```

---

**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
```

---

**KNN\_MV***KNN\_MV KEEL Preprocess Algorithm*

---

**Description**

KNN\_MV Preprocess Algorithm from KEEL.

**Usage**

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

**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
```

---

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	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::KSNN_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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.

**Examples**

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

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

#Run algorithm
```



```
#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.

### Examples

```
#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
```

---

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
```

---

LinearLMS\_R

*LinearLMS\_R KEEL Regression Algorithm*

---

### Description

LinearLMS\_R Regression Algorithm from KEEL.

### Usage

```
LinearLMS_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.

**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
```

---

loadKeelDataset	<i>Load KEEL Dataset</i>
-----------------	--------------------------

---

**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.

**Examples**

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

---

Logistic_C	<i>Logistic_C KEEL Classification Algorithm</i>
------------	---

---

**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
```

---

LVF_IEP_FS	<i>LVF_IEP_FS KEEL Preprocess Algorithm</i>
------------	---

---

**Description**

LVF\_IEP\_FS Preprocess Algorithm from KEEL.

**Usage**

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

**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)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

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"

**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
```

---

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"
pruningFactor	pruningFactor. Default value = 2
unsmoothed	unsmoothed. Default value = TRUE

**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::M5_R(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

 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	newMax. Default value = 1.0

**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::MinMax_TR(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

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.



**Examples**

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

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

#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	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 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
```

---

 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 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
```

---

MODENAR\_A

---

*MODENAR\_A KEEL Association Rules Algorithm*


---

**Description**

MODENAR\_A Association Rules Algorithm from KEEL.

**Usage**

```
MODENAR_A(dat, seed, PopulationSize, NumberofEvaluations, CrossoverrateCR,
  Thresholdforthenumberofnondominationsolutions,
  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
Thresholdforthenumberofnondominationsolutions	Threshold for the number of non-dominated solutions. Default value = 60
Thefactorofamplitudeforeachattributeofthedataset	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 algorithm

`$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)

"kloggen", Kloggen (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)

"ralambrodra rainy", Ralambrodra rainy Measure (Ralambrodra rainy, 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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- MODENAR_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("myrules")

#Save rules in PMML file
#algorithm$writePMML("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, NumberofObjectives, NumberofEvaluations, PopulationSize,
  PointCrossover, ProbabilityofCrossover, ProbabilityofMutation,
  Thefactorofamplitudeforeachattributeofthedataset)
```

### Arguments

dat	Dataset as a data.frame object
seed	seed. Default value = 1286082570
NumberofObjectives	Number of Objectives. 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
Thefactorofamplitudeforeachattributeofthedataset	The factor of amplitude for each attribute of the dataset. Default value = 2.0

### Details

`$run()` Run algorithm

`$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)

"klogen", Klogen (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)

"ralambrodraimy", Ralambrodraimy Measure (Ralambrodraimy, 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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- MOEA_Ghosh_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

MOPNAR\_A

*MOPNAR\_A KEEL Association Rules Algorithm*

---

### Description

MOPNAR\_A Association Rules Algorithm from KEEL.

**Usage**

```
MOPNAR_A(dat, seed, objectives, evaluations, parameter, weightNeighborhood,
          wprobabilitySolutionsNeighborhood, maxSolutions, probabilityMutation,
          amplitude, threshold)
```

**Arguments**

dat	Dataset as a data.frame object
seed	seed. Default value = 1286082570
objectives	objectives. Default value = 3
evaluations	evaluations. Default value = 50000
parameter	parameter. Default value = 13
weightNeighborhood	weightNeighborhood. Default value = 10
wprobabilitySolutionsNeighborhood	wprobabilitySolutionsNeighborhood. 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 algorithm

`$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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
```

```
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- MOPNAR_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("myrules")
```

---

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.

**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
```

---

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.

**Examples**

```
#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
```

---

 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
Thefactorofamplitudeforeachattributeofthedataset	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 algorithm

`$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)

"ralambrodriainy", Ralambrodriainy Measure (Ralambrodriainy, 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 iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- NICGAR_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY", "yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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

**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
```

---

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.

**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
```

---

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.

**Examples**

```
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
```

---

 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	Train dataset as a data.frame object
test	Test dataset as a data.frame object
KernelType	KernelType. Default value = 1
C	C. Default value = "RBF"
eps	eps. Default value = 1000.0
degree	degree. Default value = 0.001
gamma	gamma. Default value = 10
coef0	coef0. Default value = 0.01
nu	nu. Default value = 0.1
p	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.

**Examples**

```
#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
```

---

 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	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.

**Examples**

```
#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
```

---

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 = 0.25
itemsetsPerLeaf	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")
#data_test <- RKEEL::loadKeelDataset("iris_test")

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

PDFC\_C

*PDFC\_C KEEL Classification Algorithm*

---

**Description**

PDFC\_C Classification Algorithm from KEEL.



**Usage**

```
PDFC_C(train, test, C, d, tolerance, epsilon, PDRFtype,  
        nominal_to_binary, preprocess_type, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
C	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	preprocess_type. 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::PDFC_C(data_train, data_test)  
  
#Run algorithm  
#algorithm$run()  
  
#See results  
#algorithm$testPredictions
```

---

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	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
```

---

PNN\_C

*PNN\_C KEEL Classification Algorithm*

---

**Description**

PNN\_C Classification Algorithm from KEEL.

**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
```

---

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

**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
```

---

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.

**Examples**

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

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

#Run algorithm
```

```
#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.

### Examples

```
#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
```

---

PreprocessAlgorithm    *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 placed in 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.

### Examples

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

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

```
#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.

### Examples

```
#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
```

---

 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, enviromentSize, numParticles,
  x, c1, c2, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
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	numParticles. Default value = 100
x	x. Default value = 0.72984
c1	c1. Default value = 2.05
c2	c2. Default value = 2.05
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::PSO_ACO_C(data_train, data_test)

#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.

### Examples

```
#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
```

---

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.

### Examples

```
#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
```

---

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.

## Examples

```
#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
```

---

QAR\_CIP\_NSII\_A      *QAR\_CIP\_NSII\_A KEEL Association Rules Algorithm*

---

### Description

QAR\_CIP\_NSII\_A Association Rules Algorithm from KEEL.

### Usage

```
QAR_CIP_NSII_A(dat, seed, NumberOfObjectives, NumberOfEvaluations,
  PopulationSize, ProbabilityofMutation,
  Thefactorofamplitudeforeachattributeofthedataset, Differencethreshold)
```

### Arguments

dat	Dataset as a data.frame object
seed	seed. Default value = 1286082570
NumberOfObjectives	Number of Objectives. Default value = 3
NumberOfEvaluations	Number of Evaluations. Default value = 50000
PopulationSize	Population Size. Default value = 100
ProbabilityofMutation	Probability of Mutation. Default value = 0.1
Thefactorofamplitudeforeachattributeofthedataset	The factor of amplitude for each attribute of the dataset. Default value = 2.0
Differencethreshold	Difference threshold. Default value = 5.0

### Details

```
$run() Run algorithm

$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.

**Examples**

```
#Load KEEL dataset iris
#dat<-loadKeelDataset("iris")

#Create algorithm
#algorithm <- QAR_CIP_NSGAII_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 interest measure YuleY to set rules
#algorithm$addInterestMeasure("YuleY","yulesY")

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

#Save rules in CSV file
#algorithm$writeCSV("myrules")

#Save rules in PMML file
#algorithm$writePMML("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
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
```

---

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	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.



**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
```

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	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.

**Examples**

```
#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
```

---

read.keel	<i>Read keel dataset</i>
-----------	--------------------------

---

**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	<i>Regression Algorithm</i>
---------------------	-----------------------------

---

**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	<i>Regression Results</i>
-------------------	---------------------------

---

**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

*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
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.

### Examples

```
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
```

---

**Ripper\_C***Ripper\_C KEEL Classification Algorithm*

---

**Description**

Ripper\_C Classification Algorithm from KEEL.

**Usage**

```
Ripper_C(train, test, grow_pct, k, seed)
```

**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
k	k. 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.

**Examples**

```
#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
```

---

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
```

---

runCV                      *Run Cross-Validation*

---

**Description**

Run a cross-validation experiment

**Usage**

```
runCV(algorithm, dataset, numFolds, cores)
```

**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 (sequential 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)
```

---

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

## 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_FRNN_C <- RKEEL::FRNN_C(iris_train, iris_test)
#learner_FuzzyKNN_C <- RKEEL::FuzzyKNN_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_FRNN_C, learner_FuzzyKNN_C,
# learner_KNN_C, learner_Logistic_C, learner_LDA_C)

#Run algorithms in parallel in two cores
#par <- RKEEL::runParallel(algorithms, 2)
```

---

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

## 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_FRNN_C <- RKEEL::FRNN_C(iris_train, iris_test)
#learner_FuzzyKNN_C <- RKEEL::FuzzyKNN_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_FRNN_C, learner_FuzzyKNN_C,
# learner_KNN_C, learner_Logistic_C, learner_LDA_C)

#Run algorithms
#par <- RKEEL::runSequential(algorithms)

```

---

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.

### Examples

```

#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

```



---

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
```

---

SGA\_C

*SGA\_C KEEL Classification Algorithm*

---

**Description**

SGA\_C Classification Algorithm from KEEL.

**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	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::SGA_C(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

`Shrink_C`*Shrink\_C KEEL Classification Algorithm*

---

**Description**

Shrink\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>k</code>	k. Default value = 1
<code>distance</code>	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
```

---

`Slipper_C`*Slipper\_C KEEL Classification Algorithm*

---

**Description**

Slipper\_C Classification Algorithm from KEEL.

**Usage**

```
Slipper_C(train, test, grow_pct, numBoosting, seed)
```

**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
```

---

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)
```

**Arguments**

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

```

---

SSGA\_Integer\_knn\_FS    *SSGA\_Integer\_knn\_FS KEEL Preprocess Algorithm*

---

**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.

**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$run()

#See results
#algorithm$preprocessed_test
```

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 = 0.7
w2	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

**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$run()

#See results
#algorithm$testPredictions
```

---

 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.



**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$run()

#See results
#algorithm$testPredictions
```

---

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.

**Examples**

```
#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()
```

```
#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

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::UniformWidth_D(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

VWFuzzyKNN_C	<i>VWFuzzyKNN_C KEEL Classification Algorithm</i>
--------------	---

---

**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
```

---

WM_R	<i>WM_R KEEL Regression Algorithm</i>
------	---------------------------------------

---

**Description**

WM\_R Regression Algorithm from KEEL.

**Usage**

```
WM_R(train, test, numlabels, KB)
```

**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
```

---

writeDatFromDataframe *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

**Examples**

```
#data(iris)
#writeDatFromDataframe(iris, "iris.dat")
```

---

`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**

<code>trainData</code>	Train data as data.frame object
<code>testData</code>	Test data as data.frame object
<code>trainFileName</code>	String with the file name to store the train dataset
<code>testFileName</code>	String with the file name to store the test dataset

---

`ZScore_TR`*ZScore\_TR KEEL Preprocess Algorithm*

---

**Description**

ZScore\_TR Preprocess Algorithm from KEEL.

**Usage**

```
ZScore_TR(train, test)
```

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	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::ZScore_TR(data_train, data_test)

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

# Index

## \* association rules

- Alatasetal\_A, [8](#)
- Alcalaetal\_A, [12](#)
- Apriori\_A, [18](#)
- AssociationRulesAlgorithm, [22](#)
- EARMGA\_A, [47](#)
- Eclat\_A, [51](#)
- FPgrowth\_A, [57](#)
- FuzzyApriori\_A, [64](#)
- GAR\_A, [71](#)
- GENAR\_A, [75](#)
- GeneticFuzzyApriori\_A, [83](#)
- GeneticFuzzyAprioriDC\_A, [79](#)
- MODENAR\_A, [115](#)
- MOEA\_Ghosh\_A, [119](#)
- MOPNAR\_A, [122](#)
- NICGAR\_A, [128](#)
- QAR\_CIP\_NSGAII\_A, [148](#)

## \* classification

- AdaBoost\_I, [7](#)
- AdaBoostNC\_C, [6](#)
- ART\_C, [21](#)
- AssociativeClassificationAlgorithm, [22](#)
- BNGE\_C, [23](#)
- Bojarczuk\_GP\_C, [24](#)
- BSE\_C, [25](#)
- C45\_C, [28](#)
- C45Binarization\_C, [26](#)
- C45Rules\_C, [27](#)
- C\_SVM\_C, [42](#)
- CamNN\_C, [29](#)
- CART\_C, [29](#)
- CBA\_C, [31](#)
- CenterNN\_C, [32](#)
- CFAR\_C, [32](#)
- CFKNN\_C, [33](#)
- CHC\_C, [34](#)
- ClassificationAlgorithm, [35](#)

- ClassificationResults, [35](#)
- CMAR\_C, [37](#)
- CNN\_C, [38](#)
- CPAR\_C, [39](#)
- CPW\_C, [40](#)
- CW\_C, [41](#)
- DecrRBFN\_C, [43](#)
- Deeps\_C, [44](#)
- DSM\_C, [45](#)
- DT\_GA\_C, [46](#)
- Falco\_GP\_C, [55](#)
- FCRA\_C, [56](#)
- FRNN\_C, [61](#)
- FURIA\_C, [63](#)
- FuzzyFARCHD\_C, [67](#)
- FuzzyKNN\_C, [68](#)
- FuzzyNPC\_C, [69](#)
- GANN\_C, [70](#)
- GFS\_AdaBoost\_C, [87](#)
- GFS\_LogitBoost\_C, [91](#)
- ID3\_C, [94](#)
- IF\_KNN\_C, [95](#)
- ImbalancedClassificationAlgorithm, [97](#)
- IncrRBFN\_C, [97](#)
- JFKNN\_C, [99](#)
- Kernel\_C, [100](#)
- KNN\_C, [102](#)
- KSNN\_C, [103](#)
- KStar\_C, [104](#)
- LDA\_C, [105](#)
- LinearLMS\_C, [106](#)
- Logistic\_C, [108](#)
- MLP\_BP\_C, [112](#)
- NB\_C, [127](#)
- NM\_C, [131](#)
- NNEP\_C, [132](#)
- NU\_SVM\_C, [134](#)
- PART\_C, [136](#)

- PDFC\_C, [136](#)
- PFKNN\_C, [138](#)
- PNN\_C, [138](#)
- PolQuadraticLMS\_C, [139](#)
- PRISM\_C, [142](#)
- PSO\_ACO\_C, [144](#)
- PUBLIC\_C, [146](#)
- PW\_C, [147](#)
- QDA\_C, [151](#)
- RBFN\_C, [152](#)
- Ripper\_C, [156](#)
- RISE\_C, [157](#)
- SGA\_C, [161](#)
- Shrink\_C, [163](#)
- Slipper\_C, [163](#)
- SMO\_C, [164](#)
- Tan\_GP\_C, [167](#)
- VWFuzzyKNN\_C, [171](#)
- \* preprocess**
  - ABB\_IEP\_FS, [5](#)
  - AllKNN\_TSS, [15](#)
  - AllPossible\_MV, [16](#)
  - ANR\_F, [17](#)
  - Bayesian\_D, [23](#)
  - CleanAttributes\_TR, [36](#)
  - ClusterAnalysis\_D, [36](#)
  - DecimalScaling\_TR, [43](#)
  - ID3\_D, [94](#)
  - Ignore\_MV, [96](#)
  - IterativePartitioningFilter\_F, [98](#)
  - KMeans\_MV, [101](#)
  - KNN\_MV, [102](#)
  - LVF\_IEP\_FS, [108](#)
  - MinMax\_TR, [111](#)
  - ModelCS\_TSS, [114](#)
  - MostCommon\_MV, [126](#)
  - Nominal2Binary\_TR, [133](#)
  - POP\_TSS, [141](#)
  - PreprocessAlgorithm, [142](#)
  - Proportional\_D, [143](#)
  - PSRCG\_TSS, [145](#)
  - Relief\_FS, [155](#)
  - SaturationFilter\_F, [160](#)
  - SFS\_IEP\_FS, [161](#)
  - SSGA\_Integer\_knn\_FS, [166](#)
  - UniformFrequency\_D, [169](#)
  - UniformWidth\_D, [170](#)
  - ZScore\_TR, [173](#)
- \* regression**
  - CART\_R, [30](#)
  - EPSILON\_SVR\_R, [54](#)
  - FRSBM\_R, [62](#)
  - GFS\_GP\_R, [88](#)
  - GFS\_GSP\_R, [89](#)
  - GFS\_RB\_MF\_R, [92](#)
  - LinearLMS\_R, [106](#)
  - M5\_R, [110](#)
  - M5Rules\_R, [109](#)
  - MLP\_BP\_R, [113](#)
  - NU\_SVR\_R, [135](#)
  - PolQuadraticLMS\_R, [140](#)
  - RBFN\_R, [153](#)
  - RegressionAlgorithm, [154](#)
  - RegressionResults, [154](#)
  - Thrift\_R, [168](#)
  - WM\_R, [171](#)
- \* utils**
  - getAttributeLinesFromDataframes, [87](#)
  - hasContinuousData, [93](#)
  - hasMissingValues, [93](#)
  - isMultiClass, [98](#)
  - loadKeelDataset, [107](#)
  - read.keel, [154](#)
  - runCV, [157](#)
  - runParallel, [158](#)
  - runSequential, [159](#)
  - writeDatFromDataframe, [172](#)
  - writeDatFromDataframes, [173](#)
- ABB\_IEP\_FS, [5](#)
- AdaBoost\_I, [7](#)
- AdaBoostNC\_C, [6](#)
- Alatasetal\_A, [8](#)
- Alcalaetal\_A, [12](#)
- AllKNN\_TSS, [15](#)
- AllPossible\_MV, [16](#)
- ANR\_F, [17](#)
- Apriori\_A, [18](#)
- ART\_C, [21](#)
- AssociationRulesAlgorithm, [22](#)
- AssociativeClassificationAlgorithm, [22](#)
- Bayesian\_D, [23](#)
- BNGE\_C, [23](#)
- Bojarczuk\_GP\_C, [24](#)
- BSE\_C, [25](#)



- C45\_C, 28
- C45Binarization\_C, 26
- C45Rules\_C, 27
- C\_SVM\_C, 42
- CamNN\_C, 29
- CART\_C, 29
- CART\_R, 30
- CBA\_C, 31
- CenterNN\_C, 32
- CFAR\_C, 32
- CFKNN\_C, 33
- CHC\_C, 34
- ClassificationAlgorithm, 35
- ClassificationResults, 35
- CleanAttributes\_TR, 36
- ClusterAnalysis\_D, 36
- CMAR\_C, 37
- CNN\_C, 38
- CPAR\_C, 39
- CPW\_C, 40
- CW\_C, 41
  
- DecimalScaling\_TR, 43
- DecrRBFN\_C, 43
- Deeps\_C, 44
- DSM\_C, 45
- DT\_GA\_C, 46
  
- EARMGA\_A, 47
- EcLat\_A, 51
- EPSILON\_SVR\_R, 54
  
- Falco\_GP\_C, 55
- FCRA\_C, 56
- FPgrowth\_A, 57
- 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, 71
- GENAR\_A, 75
- GeneticFuzzyApriori\_A, 83
- GeneticFuzzyAprioriDC\_A, 79
- getAttributeLinesFromDataframes, 87
  
- GFS\_AdaBoost\_C, 87
- GFS\_GP\_R, 88
- GFS\_GSP\_R, 89
- GFS\_LogitBoost\_C, 91
- GFS\_RB\_MF\_R, 92
  
- hasContinuousData, 93
- hasMissingValues, 93
  
- ID3\_C, 94
- ID3\_D, 94
- IF\_KNN\_C, 95
- Ignore\_MV, 96
- ImbalancedClassificationAlgorithm, 97
- IncrRBFN\_C, 97
- isMultiClass, 98
- IterativePartitioningFilter\_F, 98
  
- JFKNN\_C, 99
  
- KeelAlgorithm, 100
- Kernel\_C, 100
- KMeans\_MV, 101
- KNN\_C, 102
- KNN\_MV, 102
- KSNN\_C, 103
- KStar\_C, 104
  
- LDA\_C, 105
- LinearLMS\_C, 106
- LinearLMS\_R, 106
- loadKeelDataset, 107
- Logistic\_C, 108
- LVF\_IEP\_FS, 108
  
- M5\_R, 110
- M5Rules\_R, 109
- MinMax\_TR, 111
- MLP\_BP\_C, 112
- MLP\_BP\_R, 113
- ModelCS\_TSS, 114
- MODENAR\_A, 115
- MOEA\_Ghosh\_A, 119
- MOPNAR\_A, 122
- MostCommon\_MV, 126
  
- NB\_C, 127
- NICGAR\_A, 128
- NM\_C, 131
- NNEP\_C, 132

- Nominal2Binary\_TR, 133
- NU\_SVM\_C, 134
- NU\_SVR\_R, 135
  
- PART\_C, 136
- PDFC\_C, 136
- PFKNN\_C, 138
- PNN\_C, 138
- PolQuadraticLMS\_C, 139
- PolQuadraticLMS\_R, 140
- POP\_TSS, 141
- PreprocessAlgorithm, 142
- PRISM\_C, 142
- Proportional\_D, 143
- PSO\_ACO\_C, 144
- PSRCG\_TSS, 145
- PUBLIC\_C, 146
- PW\_C, 147
  
- QAR\_CIP\_NSGAII\_A, 148
- QDA\_C, 151
  
- R6\_ABB\_IEP\_FS (ABB\_IEP\_FS), 5
- R6\_AdaBoost\_I (AdaBoost\_I), 7
- R6\_AdaBoostNC\_C (AdaBoostNC\_C), 6
- R6\_Alataetal\_A (Alataetal\_A), 8
- R6\_Alcalaetal\_A (Alcalaetal\_A), 12
- R6\_AllKNN\_TSS (AllKNN\_TSS), 15
- R6\_AllPosible\_MV (AllPosible\_MV), 16
- R6\_ANR\_F (ANR\_F), 17
- R6\_Apriori\_A (Apriori\_A), 18
- R6\_ART\_C (ART\_C), 21
- R6\_Bayesian\_D (Bayesian\_D), 23
- R6\_BNGE\_C (BNGE\_C), 23
- R6\_Bojarczuk\_GP\_C (Bojarczuk\_GP\_C), 24
- R6\_BSE\_C (BSE\_C), 25
- R6\_C45\_C (C45\_C), 28
- R6\_C45Binarization\_C  
(C45Binarization\_C), 26
- R6\_C45Rules\_C (C45Rules\_C), 27
- R6\_C\_SVM\_C (C\_SVM\_C), 42
- R6\_CamNN\_C (CamNN\_C), 29
- R6\_CART\_C (CART\_C), 29
- R6\_CART\_R (CART\_R), 30
- R6\_CBA\_C (CBA\_C), 31
- R6\_CenterNN\_C (CenterNN\_C), 32
- R6\_CFAR\_C (CFAR\_C), 32
- R6\_CFKNN\_C (CFKNN\_C), 33
- R6\_CHC\_C (CHC\_C), 34
  
- R6\_CleanAttributes\_TR  
(CleanAttributes\_TR), 36
- R6\_ClusterAnalysis\_D  
(ClusterAnalysis\_D), 36
- R6\_CMAR\_C (CMAR\_C), 37
- R6\_CNN\_C (CNN\_C), 38
- R6\_CPAR\_C (CPAR\_C), 39
- R6\_CPW\_C (CPW\_C), 40
- R6\_CW\_C (CW\_C), 41
- R6\_DecimalScaling\_TR  
(DecimalScaling\_TR), 43
- R6\_DecrRBFN\_C (DecrRBFN\_C), 43
- R6\_Deeps\_C (Deeps\_C), 44
- R6\_DSM\_C (DSM\_C), 45
- R6\_DT\_GA\_C (DT\_GA\_C), 46
- R6\_EARMGA\_A (EARMGA\_A), 47
- R6\_Eclat\_A (Eclat\_A), 51
- R6\_EPSILON\_SVR\_R (EPSILON\_SVR\_R), 54
- R6\_Falco\_GP\_C (Falco\_GP\_C), 55
- R6\_FCRA\_C (FCRA\_C), 56
- R6\_FPgrowth\_A (FPgrowth\_A), 57
- R6\_FRNN\_C (FRNN\_C), 61
- R6\_FRSBM\_R (FRSBM\_R), 62
- R6\_FURIA\_C (FURIA\_C), 63
- R6\_FuzzyApriori\_A (FuzzyApriori\_A), 64
- R6\_FuzzyFARCHD\_C (FuzzyFARCHD\_C), 67
- R6\_FuzzyKNN\_C (FuzzyKNN\_C), 68
- R6\_FuzzyNPC\_C (FuzzyNPC\_C), 69
- R6\_GANN\_C (GANN\_C), 70
- R6\_GAR\_A (GAR\_A), 71
- R6\_GENAR\_A (GENAR\_A), 75
- R6\_GeneticFuzzyApriori\_A  
(GeneticFuzzyApriori\_A), 83
- R6\_GeneticFuzzyAprioriDC\_A  
(GeneticFuzzyAprioriDC\_A), 79
- R6\_GFS\_AdaBoost\_C (GFS\_AdaBoost\_C), 87
- R6\_GFS\_GP\_R (GFS\_GP\_R), 88
- R6\_GFS\_GSP\_R (GFS\_GSP\_R), 89
- R6\_GFS\_LogitBoost\_C (GFS\_LogitBoost\_C),  
91
- R6\_GFS\_RB\_MF\_R (GFS\_RB\_MF\_R), 92
- R6\_ID3\_C (ID3\_C), 94
- R6\_ID3\_D (ID3\_D), 94
- R6\_IF\_KNN\_C (IF\_KNN\_C), 95
- R6\_Ignore\_MV (Ignore\_MV), 96
- R6\_IncrRBFN\_C (IncrRBFN\_C), 97
- R6\_IterativePartitioningFilter\_F  
(IterativePartitioningFilter\_F),

- 98
- R6\_JFKNN\_C (JFKNN\_C), 99
- R6\_Kernel\_C (Kernel\_C), 100
- R6\_KMeans\_MV (KMeans\_MV), 101
- R6\_KNN\_C (KNN\_C), 102
- R6\_KNN\_MV (KNN\_MV), 102
- R6\_KSNN\_C (KSNN\_C), 103
- R6\_KStar\_C (KStar\_C), 104
- R6\_LDA\_C (LDA\_C), 105
- R6\_LinearLMS\_C (LinearLMS\_C), 106
- R6\_LinearLMS\_R (LinearLMS\_R), 106
- R6\_Logistic\_C (Logistic\_C), 108
- R6\_LVF\_IEP\_FS (LVF\_IEP\_FS), 108
- R6\_M5\_R (M5\_R), 110
- R6\_M5Rules\_R (M5Rules\_R), 109
- R6\_MinMax\_TR (MinMax\_TR), 111
- R6\_MLP\_BP\_C (MLP\_BP\_C), 112
- R6\_MLP\_BP\_R (MLP\_BP\_R), 113
- R6\_ModelCS\_TSS (ModelCS\_TSS), 114
- R6\_MODENAR\_A (MODENAR\_A), 115
- R6\_MOEA\_Ghosh\_A (MOEA\_Ghosh\_A), 119
- R6\_MOPNAR\_A (MOPNAR\_A), 122
- R6\_MostCommon\_MV (MostCommon\_MV), 126
- R6\_NB\_C (NB\_C), 127
- R6\_NICGAR\_A (NICGAR\_A), 128
- R6\_NM\_C (NM\_C), 131
- R6\_NNEP\_C (NNEP\_C), 132
- R6\_Nominal2Binary\_TR  
(Nominal2Binary\_TR), 133
- R6\_NU\_SVM\_C (NU\_SVM\_C), 134
- R6\_NU\_SVR\_R (NU\_SVR\_R), 135
- R6\_PART\_C (PART\_C), 136
- R6\_PDFC\_C (PDFC\_C), 136
- R6\_PFKNN\_C (PFKNN\_C), 138
- R6\_PNN\_C (PNN\_C), 138
- R6\_PolQuadraticLMS\_C  
(PolQuadraticLMS\_C), 139
- R6\_PolQuadraticLMS\_R  
(PolQuadraticLMS\_R), 140
- R6\_POP\_TSS (POP\_TSS), 141
- R6\_PRISM\_C (PRISM\_C), 142
- R6\_Proportional\_D (Proportional\_D), 143
- R6\_PSO\_ACO\_C (PSO\_ACO\_C), 144
- R6\_PSRG\_TSS (PSRCG\_TSS), 145
- R6\_PUBLIC\_C (PUBLIC\_C), 146
- R6\_PW\_C (PW\_C), 147
- R6\_QAR\_CIP\_NSGAII\_A (QAR\_CIP\_NSGAII\_A),  
148
- R6\_QDA\_C (QDA\_C), 151
- R6\_RBFN\_C (RBFN\_C), 152
- R6\_RBFN\_R (RBFN\_R), 153
- R6\_Relief\_FS (Relief\_FS), 155
- R6\_Ripper\_C (Ripper\_C), 156
- R6\_RISE\_C (RISE\_C), 157
- R6\_SaturationFilter\_F  
(SaturationFilter\_F), 160
- R6\_SFS\_IEP\_FS (SFS\_IEP\_FS), 161
- R6\_SGA\_C (SGA\_C), 161
- R6\_Shrink\_C (Shrink\_C), 163
- R6\_Slipper\_C (Slipper\_C), 163
- R6\_SMO\_C (SMO\_C), 164
- R6\_SSGA\_Integer\_knn\_FS  
(SSGA\_Integer\_knn\_FS), 166
- R6\_Tan\_GP\_C (Tan\_GP\_C), 167
- R6\_Thrift\_R (Thrift\_R), 168
- R6\_UniformFrequency\_D  
(UniformFrequency\_D), 169
- R6\_UniformWidth\_D (UniformWidth\_D), 170
- R6\_VWFuzzyKNN\_C (VWFuzzyKNN\_C), 171
- R6\_WM\_R (WM\_R), 171
- R6\_ZScore\_TR (ZScore\_TR), 173
- RBFN\_C, 152
- RBFN\_R, 153
- read.keel, 154
- RegressionAlgorithm, 154
- RegressionResults, 154
- Relief\_FS, 155
- Ripper\_C, 156
- RISE\_C, 157
- runCV, 157
- runParallel, 158
- runSequential, 159
- SaturationFilter\_F, 160
- SFS\_IEP\_FS, 161
- SGA\_C, 161
- Shrink\_C, 163
- Slipper\_C, 163
- SMO\_C, 164
- SSGA\_Integer\_knn\_FS, 166
- Tan\_GP\_C, 167
- Thrift\_R, 168
- UniformFrequency\_D, 169
- UniformWidth\_D, 170
- VWFuzzyKNN\_C, 171

WM\_R, [171](#)

writeDatFromDataframe, [172](#)

writeDatFromDataframes, [173](#)

ZScore\_TR, [173](#)