Package: RKEEL (via r-universe)

September 12, 2024

Type Package

Title Using 'KEEL' in R Code

Version 1.3.4

Depends R (>= 3.4.0)

Date 2023-09-07

Author Jose M. Moyano [aut, cre], Luciano Sanchez [aut], Oliver Sanchez [ctb], Jesus Alcala-Fernandez [ctb]

Maintainer Jose M. Moyano <jmoyano1@us.es>

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, RKEELdata (>= 1.0.5), pmml, arules, Matrix, rJava, openssl, downloader

NeedsCompilation no

Date/Publication 2023-09-14 18:50:08 UTC

Repository https://i02momuj.r-universe.dev

RemoteUrl https://github.com/cran/RKEEL

RemoteRef HEAD

RemoteSha 2aaf54ca2bd7890405775905887 0a515a3d83437

Contents

`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
```

AdaBoost_I *AdaBoost_I KEEL Imbalanced Classification Algorithm*

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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

`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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

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

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

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

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

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

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::AllPosible_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("zoo")
data_test <- RKEEL::loadKeelDataset("zoo")

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test
```

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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

#Create algorithm
```

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

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

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

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

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

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

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

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

`AssociativeClassificationAlgorithm`

Associative Classification Algorithm

Description

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

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

Bayesian_D *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, maxDepth=3)

#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 placen ind a temporary random directory and then removed. If indicated, the experiment folder is not removed. The expUniqueName parameter indicates the name of the experiment folder. If not indicated, it is a random name. If indicated, ensure that the name is unique in the previously indicated folder. The javaOptions parameter indicates, if wanted, extra parameters to the java command line, as for example the maximum memory allowed by java.

ClassificationResults *Classification Results*

Description

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

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
```

---

downloadFromMirror *Download file from a mirror*

Description

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

Usage

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

Arguments

`mirror`	URL from which to download the file.
`file_path`	Path or folder where the downloaded file will be stored.
`md5_sum`	md5 checksum string corresponding to the file to download. The method will check that the downloaded file checksum and the md5_sum parameter match.

Value

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

Examples

```
# Download RKEELjars file
dCode = RKEEL::downloadFromMirror("https://personal.us.es/jmoyano1/RKEELjars_1.1.zip",
  downloadedJarFile, md5_sum)

# Check if the download was successful
if(dCode<0){
  print('There was an error during the download.')
}
```

DSM_C *DSM_C KEEL Classification Algorithm*

Description

DSM_C Classification Algorithm from KEEL.

Usage

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

Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
iterations	iterations. Default value = 100
percentage	percentage. Default value = 10
alpha_0	alpha_0. Default value = 0.1
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

Value

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

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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

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

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

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

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

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

Eclat_A *Eclat_A KEEL Association Rules Algorithm*

Description

Eclat_A Association Rules Algorithm from KEEL.

Usage

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

Arguments

`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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

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

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

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

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

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

EPSILON_SVR_R *EPSILON_SVR_R KEEL Regression Algorithm*

Description

EPSILON_SVR_R Regression Algorithm from KEEL.

Usage

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

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, generations=10, pop_size=10)

#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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

FRNN_C *FRNN_C KEEL Classification Algorithm*

Description

FRNN_C Classification Algorithm from KEEL.

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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

FuzzyFARCHD_C *FuzzyFARCHD_C KEEL Classification Algorithm*

Description

FuzzyFARCHD_C Classification Algorithm from KEEL.

Usage

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

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, hidden_layers=1,
  hidden_nodes=5, max_generations=5)

#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

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

Details

`$run()` Run algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

GENAR_A *GENAR_A 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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

GeneticFuzzyAprioriDC_A

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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

#Create algorithm
algorithm <- RKEEL::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(paste0(tempdir(), "/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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#Rules in format arules
algorithm$rules

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

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

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

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

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

`getAttributeLinesFromDataframes` *Get attribute lines from data.frames*

Description

Method for getting the attribute lines from data.frame objects

Usage

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

Arguments

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

Value

Returns a list with the attribute names and types

Examples

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

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

getExePath *Get jar executable files Path*

Description

Method for knowing the KEEL .jar files path.

Usage

```
getExePath()
```

Value

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

Examples

```
getExePath()
```

getJarList *Get a list with all RKEEL algorithm jars*

Description

Method that returns a list with the jar names from RKEEL

Usage

```
getJarList()
```

Value

Returns a list with the jar names from RKEEL.

Examples

```
getJarList()
```

getJarPath *Get RunKeel.jar Path*

Description

Method for knowing the RunKeel.jar path.

Usage

```
getJarPath()
```

Value

Returns a string with the RunKeel.jar path.

Examples

```
getJarPath()
```

GFS_AdaBoost_C *GFS_AdaBoost_C KEEL Classification Algorithm*

Description

GFS_AdaBoost_C Classification Algorithm from KEEL.

Usage

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

Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
numLabels	numLabels. Default value = 3
numRules	numRules. Default value = 8
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

Value

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

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, probcrossga, probmutaga,
   lenchaingap, maxtreeheight, seed)
```

Arguments

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

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
deltafitsap	deltafitsap. Default value = 0.5
p0sap	p0sap. Default value = 0.5
p1sap	p1sap. Default value = 0.5
amplMut	amplMut. Default value = 0.1
nsubsap	nsubsap. Default value = 10
probOptimLocal	probOptimLocal. Default value = 0.00
numOptimLocal	numOptimLocal. Default value = 0
idOptimLocal	idOptimLocal. Default value = 0
probcrossga	probcrossga. Default value = 0.5
probmutaga	probmutaga. Default value = 0.5
lenchaingap	lenchaingap. Default value = 10
maxtreeheight	maxtreeheight. Default value = 8
numItera	numItera. Default value = 10000
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_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 *Has Continuous Data*

Description

Method for check if a dataset has continuous data

Usage

```
hasContinuousData(data)
```

Arguments

data	Dataset as data.frame

Value

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

Examples

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

hasMissingValues *Has Missing Values*

Description

Method for check if a dataset has missing values

Usage

```
hasMissingValues(data)
```

Arguments

data	Dataset as data.frame

Value

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

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 *Is Multi-class*

Description

Method for check if a dataset is multi-class

Usage

```
isMultiClass(data)
```

Arguments

data	Dataset as data.frame

Value

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

Examples

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

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` *Keel Algorithm*

Description

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

`Kernel_C` *Kernel_C KEEL Classification Algorithm*

Description

Kernel_C Classification Algorithm from KEEL.

Usage

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

Arguments

`train`	Train dataset as a data.frame object
`test`	Test dataset as a data.frame object
`sigma`	sigma. Default value = 0.01
`seed`	Seed for random numbers. If it is not assigned a value, the seed will be a random number

Value

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

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

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
k	k. Default value = 10
error	error. Default value = 100
iterations	iterations. Default value = 100
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

Value

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

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 *Load KEEL Dataset*

Description

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

Usage

```
loadKeelDataset(dataName)
```

Arguments

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

Value

Returns a data.frame with the KEEL dataset.

Examples

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

Logistic_C *Logistic_C KEEL Classification Algorithm*

Description

Logistic_C Classification Algorithm from KEEL.

Usage

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

Arguments

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

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

LVF_IEP_FS *LVF_IEP_FS KEEL Preprocess Algorithm*

Description

LVF_IEP_FS Preprocess Algorithm from KEEL.

Usage

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

Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
paramKNN	paramKNN. Default value = 1
maxLoops	maxLoops. Default value = 770
inconAllow	inconAllow. Default value = 0
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

Value

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

Examples

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

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

#Run algorithm
algorithm$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,
  Thresholdforthenumberofnondominatedsolutions,
  Thefactorofamplitudeforeachattributeofthedataset, WeightforSupport,
  WeightforConfidence, WeightforComprehensibility,
  WeightforAmplitudeoftheIntervals)
```

Arguments

`dat`	Dataset as a data.frame object
`seed`	seed. Default value = 1286082570
`PopulationSize`	Population Size. Default value = 100
`NumberofEvaluations`	Number of Evaluations. Default value = 50000
`CrossoverrateCR`	Crossover rate CR. Default value = 0.3
`Thresholdforthenumberofnondominatedsolutions`	Threshold for the number of non-dominated solutions. Default value = 60
`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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

#Create algorithm
algorithm <- RKEEL::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(paste0(tempdir(), "/myrules"))
```

MOEA_Ghosh_A *MOEA_Ghosh_A KEEL Association Rules Algorithm*

Description

MOEA_Ghosh_A Association Rules Algorithm from KEEL.

Usage

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

Arguments

`dat`	Dataset as a data.frame object
`seed`	seed. Default value = 1286082570
`NumberofObjetives`	Number of Objetives. Default value = 3
`NumberofEvaluations`	Number of Evaluations. Default value = 50000
`PopulationSize`	Population Size. Default value = 100
`PointCrossover`	Point Crossover. Default value = 2
`ProbabilityofCrossover`	Probability of Crossover. Default value = 0.8
`ProbabilityofMutation`	Probability of Mutation. Default value = 0.02
`Thefactorofamplitudeforeachattributeofthedataset`	The factor of amplitude for each attribute of the dataset. Default value = 2.0

Details

`$run()` Run algorith

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

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

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

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

`$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
dat<-RKEEL::loadKeelDataset("car")

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

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

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

MOPNAR_A *MOPNAR_A KEEL Association Rules Algorithm*

Description

MOPNAR_A Association Rules Algorithm from KEEL.

Usage

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

Arguments

dat	Dataset as a data.frame object
seed	seed. Default value = 1286082570
objetives	objetives. Default value = 3
evaluations	evaluations. Default value = 50000
parameter	parameter. Default value = 13
weightNeighborhood	weightNeighborhood. Default value = 10
wrobabilitySolutionsNeighborhood	wrobabilitySolutionsNeighborhood. Default value = 0.9
maxSolutions	maxSolutions. Default value = 2
probabilityMutation	probabilityMutation. Default value = 0.1
amplitude	amplitude. Default value = 2.0
threshold	threshold. Default value = 5.0

Details

`$run()` Run algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

`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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

NM_C *NM_C KEEL Classification Algorithm*

Description

NM_C Classification Algorithm from KEEL.

Usage

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

Arguments

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

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

PRISM_C *PRISM_C KEEL Classification Algorithm*

Description

PRISM_C Classification Algorithm from KEEL.

Usage

```
PRISM_C(train, test, seed)
```

Arguments

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

Value

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

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,
  max_iterations_without_converge=2, numParticles=5)

#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_NSGAII_A *QAR_CIP_NSGAII_A KEEL Association Rules Algorithm*

Description

QAR_CIP_NSGAII_A Association Rules Algorithm from KEEL.

Usage

```
QAR_CIP_NSGAII_A(dat, seed, NumberofObjetives, NumberofEvaluations,
  PopulationSize, ProbabilityofMutation,
  Thefactorofamplitudeforeachattributeofthedataset, Differencethreshold)
```

Arguments

`dat`	Dataset as a data.frame object
`seed`	seed. Default value = 1286082570
`NumberofObjetives`	Number of Objetives. Default value = 3
`NumberofEvaluations`	Number of Evaluations. Default value = 50000
`PopulationSize`	Population Size. Default value = 100
`ProbabilityofMutation`	Probability of Mutation. Default value = 0.1
`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 algorith

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

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

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

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

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

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

"allConfidence" (Omiencinski, 2003)

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

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

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

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

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

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

"collectiveStrength"

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

"conviction" (Brin et al. 1997)

"cosine" (Tan et al., 2004)

"coverage", cover, LHS-support

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999)

"casualConfidence", casual confidence (Kodratoff, 1999)

"casualSupport", casual support (Kodratoff, 1999)

"counterexample", example and counterexample rate

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999)

"doc", difference of confidence (Hofmann and Wilhelm, 2001)

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007)

"gini", Gini index (Tan et al., 2004)

"hyperLift" (Hahsler and Hornik, 2007)

"hyperConfidence" (Hahsler and Hornik, 2007)

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010)

"implicationIndex", implication index (Gras, 1996)

"improvement" (Bayardo et al., 2000)

"jaccard", Jaccard coefficient (Tan and Kumar, 2000)

"jMeasure", J-measure, J (Smyth and Goodman, 1991)

"kappa" (Tan and Kumar, 2000)

"klosgen", Klosgen (Tan and Kumar, 2000)

"kulczynski" (Wu, Chen and Han, 2007; Kulczynski, 1927)

"lambda", Goodman-Kruskal lambda, predictive association (Tan and Kumar, 2000)

"laplace", L (Tan and Kumar 2000)

"leastContradiction", least contradiction (Aze and Kodratoff, 2004

"lerman", Lerman similarity (Lerman, 1981)

"leverage", PS (Piatetsky-Shapiro 1991)

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

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

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

"ralambrodrainy", Ralambrodrainy Measure (Ralambrodrainy, 1991)

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

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

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

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

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

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

For more information see ?arules::interestMeasure

Value

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

Examples

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

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

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

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

QDA_C *QDA_C KEEL Classification Algorithm*

Description

QDA_C Classification Algorithm from KEEL.

Usage

```
QDA_C(train, test, seed)
```

Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
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 *Read keel dataset*

Description

Method for read datasets in .dat KEEL format

Usage

```
read.keel(file)
```

Arguments

file	File containing the dataset to be read. It must be in KEEL .dat format.

Value

Returns a data.frame object with the dataset

RegressionAlgorithm *Regression Algorithm*

Description

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

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

RegressionResults *Regression Results*

Description

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

Relief_FS *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_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)

#Create list
algorithms <- list(learner_C45_C, learner_KNN_C, learner_Logistic_C,
  learner_LDA_C)

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

---

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_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)
```

```
#Create list
algorithms <- list(learner_C45_C, learner_KNN_C, learner_Logistic_C,
  learner_LDA_C)

#Run algorithms
seq <- 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

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::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 *VWFuzzyKNN_C KEEL Classification Algorithm*

Description

VWFuzzyKNN_C Classification Algorithm from KEEL.

Usage

```
VWFuzzyKNN_C(train, test, k, init_k)
```

Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
k	k. Default value = 3
init_k	init_k. Default value = 3

Value

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

Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

WM_R *WM_R KEEL Regression Algorithm*

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, paste0(tempdir(), "/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

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

ZScore_TR *ZScore_TR KEEL Preprocess Algorithm*

Description

ZScore_TR Preprocess Algorithm from KEEL.

Usage

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

Arguments

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

Value

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

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