

# Package ‘RKEEL’

January 31, 2016

**Type** Package

**Title** Using Keel in R Code

**Version** 1.1.5

**Depends** R (>= 3.0)

**Date** 2016-01-31

**Author** Jose Moyano [aut, cre], Luciano Sanchez Ramos [aut]

**Maintainer** Jose Moyano <i02momuj@uco.es>

## Description

KEEL is a popular Java software for a large number of different knowledge data discovery tasks. This package takes the advantages of KEEL and R, allowing to use KEEL algorithms in simple R code. The implemented R code layer between R and KEEL makes easy both using KEEL algorithms in R as implementing new algorithms for 'RKEEL' in a very simple way. It includes more than 100 algorithms for classification, regression and preprocess, which allows a more complete experimentation process. For more information about KEEL, see <<http://www.keel.es/>>.

**SystemRequirements** Java (>= 7.0)

**License** GPL

**Imports** R6, XML, doParallel, foreach, gdata, RKEELjars, RKEELdata

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2016-01-31 16:36:42

## R topics documented:

ABB_IEP_FS . . . . .	4
AdaBoostNC_C . . . . .	5
AllKNN_TSS . . . . .	6
AllPosible_MV . . . . .	7
ANR_F . . . . .	8
ART_C . . . . .	8

AssociativeClassificationAlgorithm	9
Bayesian_D	9
BNGE_C	10
Bojarczuk_GP_C	11
BSE_C	12
C45Binarization_C	12
C45Rules_C	13
C45_C	14
CamNN_C	15
CART_C	16
CART_R	17
CenterNN_C	17
CFAR_C	18
CFKNN_C	19
CHC_C	20
ClassificationAlgorithm	21
ClassificationResults	21
CleanAttributes_TR	21
ClusterAnalysis_D	22
CNN_C	23
CPW_C	24
CW_C	25
C_SVM_C	26
DecimalScaling_TR	27
DecrRBFN_C	27
Deeps_C	28
DSM_C	29
DT_GA_C	30
EPSILON_SVR_R	31
Falco_GP_C	32
FCRA_C	33
FRNN_C	34
FRSBM_R	35
FURIA_C	36
FuzzyFARCHD_C	37
FuzzyKNN_C	38
FuzzyNPC_C	39
GANN_C	39
getAttributeLinesFromDataframes	41
GFS_AdaBoost_C	42
GFS_GP_R	43
GFS_GSP_R	44
GFS_LogitBoost_C	45
GFS_RB_MF_R	46
hasContinuousData	47
hasMissingValues	48
ID3_C	48
ID3_D	49

IF_KNN_C . . . . .	50
Ignore_MV . . . . .	51
IncrRBFN_C . . . . .	51
isMultiClass . . . . .	52
IterativePartitioningFilter_F . . . . .	53
JFKNN_C . . . . .	54
KeelAlgorithm . . . . .	54
Kernel_C . . . . .	55
KMeans_MV . . . . .	55
KNN_C . . . . .	56
KNN_MV . . . . .	57
KSNN_C . . . . .	58
KStar_C . . . . .	59
LDA_C . . . . .	60
LinearLMS_C . . . . .	60
LinearLMS_R . . . . .	61
loadKeelDataset . . . . .	62
Logistic_C . . . . .	63
LVF_IEP_FS . . . . .	63
M5Rules_R . . . . .	64
M5_R . . . . .	65
MinMax_TR . . . . .	66
MLP_BP_C . . . . .	67
MLP_BP_R . . . . .	68
ModelCS_TSS . . . . .	69
MostCommon_MV . . . . .	70
NB_C . . . . .	71
NM_C . . . . .	71
NNEP_C . . . . .	72
Nominal2Binary_TR . . . . .	73
NU_SVM_C . . . . .	74
NU_SVR_R . . . . .	75
PART_C . . . . .	76
PDFC_C . . . . .	76
PFKNN_C . . . . .	78
PNN_C . . . . .	78
PolQuadraticLMS_C . . . . .	79
PolQuadraticLMS_R . . . . .	80
POP_TSS . . . . .	81
PreprocessAlgorithm . . . . .	81
PRISM_C . . . . .	82
Proportional_D . . . . .	82
PSO_ACO_C . . . . .	83
PSRCG_TSS . . . . .	84
PUBLIC_C . . . . .	85
PW_C . . . . .	86
QDA_C . . . . .	87
RBFN_C . . . . .	87

RBFN_R . . . . .	88
read.keel . . . . .	89
RegressionAlgorithm . . . . .	89
RegressionResults . . . . .	90
Relief_FS . . . . .	90
Ripper_C . . . . .	91
RISE_C . . . . .	92
runParallel . . . . .	92
runSequential . . . . .	93
SaturationFilter_F . . . . .	94
SFS_IEP_FS . . . . .	95
SGA_C . . . . .	96
Shrink_C . . . . .	97
Slipper_C . . . . .	97
SMO_C . . . . .	98
SSGA_Integer_knn_FS . . . . .	100
Tan_GP_C . . . . .	101
Thrift_R . . . . .	102
UniformFrequency_D . . . . .	103
UniformWidth_D . . . . .	104
VWFuzzyKNN_C . . . . .	105
WM_R . . . . .	105
writeDatFromDataframe . . . . .	106
writeDatFromDataframes . . . . .	107
ZScore_TR . . . . .	107

<b>Index</b>	<b>109</b>
--------------	------------

---

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

---

## 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
```

---

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

 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
```

---

AssociativeClassificationAlgorithm

*Associative Classification Algorithm*


---

**Description**

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

---

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

---

**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	<i>C45Binarization_C KEEL Classification Algorithm</i>
-------------------	--

---

**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
itemssetsPerLeaf	itemssetsPerLeaf. 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	<i>C45_C KEEL Classification Algorithm</i>
-------	--

---

Description

C45\_C Classification Algorithm from KEEL.

Usage

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

Arguments

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

CamNN\_C

---

*CamNN\_C KEEL Classification Algorithm*


---

**Description**

CamNN\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()
```

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

---

CART\_C

---

*CART\_C KEEL Classification Algorithm*


---

### Description

CART\_C Classification Algorithm from KEEL.

### Usage

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

### Arguments

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

### Value

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

### Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```



CART\_R

*CART\_R KEEL Regression Algorithm***Description**

CART\_R Regression Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

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	<i>Classification Algorithm</i>
-------------------------	---------------------------------

---

Description

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

---

ClassificationResults	<i>Classification Results</i>
-----------------------	-------------------------------

---

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

---

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
```

---

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
```

---

**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	<i>DecimalScaling_TR KEEL Preprocess Algorithm</i>
-------------------	--

---

**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	<i>DecrRBFN_C KEEL Classification Algorithm</i>
------------	---

---

**Description**

DecrRBFN\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
percent	percent. Default value = 0.1
num_neurons_ini	num_neurons_ini. Default value = 20
alfa	alfa. Default value = 0.3
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

Deeps\_C

---

*Deeps\_C KEEL Classification Algorithm*


---

**Description**

Deeps\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
beta	beta. Default value = 0.12

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

DSM\_C

---

*DSM\_C KEEL Classification Algorithm*


---

**Description**

DSM\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

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

```

EPSILON\_SVR\_R

*EPSILON\_SVR\_R KEEL Regression Algorithm***Description**

EPSILON\_SVR\_R Regression Algorithm from KEEL.

**Usage**

```

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

```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
KernelType	KernelType. Default value = "RBF"
C	C. Default value = 100.0
eps	eps. Default value = 0.001
degree	degree. Default value = 3
gamma	gamma. Default value = 0.01
coef0	coef0. Default value = 0.0
nu	nu. Default value = 0.5
p	p. Default value = 1.0
shrinking	shrinking. Default value = 0
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

Falco\_GP\_C

*Falco\_GP\_C KEEL Classification Algorithm***Description**

Falco\_GP\_C Classification Algorithm from KEEL.

**Usage**

```

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

```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
population_size	population_size. Default value = 200
max_generations	max_generations. Default value = 200
max_deriv_size	max_deriv_size. Default value = 20
rec_prob	rec_prob. Default value = 0.8
mut_prob	mut_prob. Default value = 0.1
copy_prob	copy_prob. Default value = 0.01
alpha	alpha. Default value = 0.9
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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



**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

FCRA\_C

*FCRA\_C KEEL Classification Algorithm***Description**

FCRA\_C Classification Algorithm from KEEL.

**Usage**

```

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

```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
generations	generations. Default value = 50
pop_size	pop_size. Default value = 30
length_S_C	length_S_C. Default value = 10
WCAR	WCAR. Default value = 10.0
WV	WV. Default value = 1.0
crossover_prob	crossover_prob. Default value = 1.0
mut_prob	mut_prob. Default value = 0.01
n1	n1. Default value = 0.001
n2	n2. Default value = 0.1
max_iter	max_iter. Default value = 100
linguistic_values	linguistic_values. Default value = 5
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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
```

FuzzyFARCHD\_C

*FuzzyFARCHD\_C KEEL Classification Algorithm***Description**

FuzzyFARCHD\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
linguistic_values	linguistic_values. Default value = 5
min_support	min_support. Default value = 0.05
max_confidence	max_confidence. Default value = 0.8
depth_max	depth_max. Default value = 3
K	K. Default value = 2
max_evaluations	max_evaluations. Default value = 15000
pop_size	pop_size. Default value = 50
alpha	alpha. Default value = 0.15
bits_per_gen	bits_per_gen. Default value = 30
inference_type	inference_type. Default value = 1
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
```

```

algorithm$run()

#See results
algorithm$testPredictions

```

---

FuzzyKNN\_C

*FuzzyKNN\_C KEEL Classification Algorithm*


---

### Description

FuzzyKNN\_C Classification Algorithm from KEEL.

### Usage

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

### Arguments

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
k	k. Default value = 3
M	M. Default value = 2.0
initialization	initialization. Default value = "CRISP"
init_k	init_k. Default value = 3

### Value

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

### Examples

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

---

FuzzyNPC\_C

*FuzzyNPC\_C KEEL Classification Algorithm*


---

**Description**

FuzzyNPC\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
M	M. Default value = 2.0

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

GANN\_C

*GANN\_C KEEL Classification Algorithm*


---

**Description**

GANN\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
hidden_layers	hidden_layers. Default value = 2
hidden_nodes	hidden_nodes. Default value = 15
transfer	transfer. Default value = "Htan"
eta	eta. Default value = 0.15
alpha	alpha. Default value = 0.1
lambda	lambda. Default value = 0.0
test_data	test_data. Default value = TRUE
validation_data	validation_data. Default value = FALSE
cross_validation	cross_validation. Default value = FALSE
BP_cycles	BP_cycles. Default value = 10000
improve	improve. Default value = 0.01
tipify_inputs	tipify_inputs. Default value = TRUE
save_all	save_all. Default value = FALSE
elite	elite. Default value = 0.1
num_individuals	num_individuals. Default value = 100
w_range	w_range. Default value = 5.0
connectivity	connectivity. Default value = 0.5
P_bp	P_bp. Default value = 0.25
P_param	P_param. Default value = 0.1
P_struct	P_struct. Default value = 0.1
max_generations	max_generations. Default value = 100
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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



**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$testPredictions
```

---

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

---

**Description**

Method for getting the attribute lines from data.frame objects

**Usage**

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

**Arguments**

<code>trainData</code>	Train dataset as data.frame
<code>testData</code>	Test dataset as data.frame

**Value**

Returns a list with the attribute names and types

**Examples**

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

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

---

GFS\_AdaBoost\_C

*GFS\_AdaBoost\_C KEEL Classification Algorithm*


---

## Description

GFS\_AdaBoost\_C Classification Algorithm from KEEL.

## Usage

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

## Arguments

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

## Value

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

## Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

GFS\_GP\_R

*GFS\_GP\_R KEEL Regression Algorithm***Description**

GFS\_GP\_R Regression Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
numLabels	numLabels. Default value = 3
numRules	numRules. Default value = 8
popSize	popSize. Default value = 30
numisland	numisland. Default value = 2
steady	steady. Default value = 1
numIter	numIter. Default value = 100
tourSize	tourSize. Default value = 4
mutProb	mutProb. Default value = 0.01
aplMut	aplMut. Default value = 0.1
probMigra	probMigra. Default value = 0.001
probOptimLocal	probOptimLocal. Default value = 0.00
numOptimLocal	numOptimLocal. Default value = 0
idOptimLocal	idOptimLocal. Default value = 0
nichinggap	nichinggap. Default value = 0
maxindniche	maxindniche. Default value = 8
probintraniche	probintraniche. Default value = 0.75
probcrosssga	probcrosssga. 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, probcrosssga, 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	<i>Has Continuous Data</i>
-------------------	----------------------------

---

**Description**

Method for check if a dataset has continuous data

**Usage**

```
hasContinuousData(data)
```

**Arguments**

data	Dataset as data.frame
------	-----------------------

**Value**

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

**Examples**

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

---

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

---

**Description**

Method for check if a dataset has missing values

**Usage**

```
hasMissingValues(data)
```

**Arguments**

data	Dataset as data.frame
------	-----------------------

**Value**

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

**Examples**

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

---

ID3_C	<i>ID3_C KEEL Classification Algorithm</i>
-------	--

---

**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
```

---

IncrRBFN\_C

---

*IncrRBFN\_C KEEL Classification Algorithm*


---

**Description**

IncrRBFN\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

**Description**

Method for check if a dataset is multi-class

**Usage**

```
isMultiClass(data)
```

**Arguments**

data	Dataset as data.frame
------	-----------------------

**Value**

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

**Examples**

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

---

IterativePartitioningFilter\_F

*IterativePartitioningFilter\_F KEEL Preprocess Algorithm*


---

**Description**

IterativePartitioningFilter\_F Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
numPartitions	numPartitions. Default value = 5
filterType	filterType. Default value = "consensus"
confidence	confidence. Default value = 0.25
itemsetsPerLeaf	itemsetsPerLeaf. Default value = 2
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test
```

---

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

---

**Description**

JFKNN\_C Classification Algorithm from KEEL.

**Usage**

JFKNN\_C(train, test)

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

**Description**

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

Kernel\_C

*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**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>selection_method</code>	selection_method. Default value = "Fixed"
<code>blend</code>	blend. Default value = 0.2
<code>seed</code>	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

LDA\_C

*LDA\_C KEEL Classification Algorithm***Description**

LDA\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

LinearLMS\_C

*LinearLMS\_C KEEL Classification Algorithm***Description**

LinearLMS\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

LinearLMS\_R

---

*LinearLMS\_R KEEL Regression Algorithm*


---

**Description**

LinearLMS\_R Regression Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

### Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

### Description

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

### Usage

```
loadKeelDataset(dataName)
```

### Arguments

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

### Value

Returns a `data.frame` with the KEEL dataset.

### Examples

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

---

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

---

**Description**

Logistic\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

**Description**

LVF\_IEP\_FS Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test
```

---

M5Rules\_R

---

*M5Rules\_R KEEL Regression Algorithm*


---

**Description**

M5Rules\_R Regression Algorithm from KEEL.

**Usage**

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

**Arguments**

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



**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

M5\_R

---

*M5\_R KEEL Regression Algorithm*


---

**Description**

M5\_R Regression Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
type	type. Default value = "m"
pruningFactor	pruningFactor. Default value = 2
unsmoothed	unsmoothed. Default value = TRUE

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

---

MinMax\_TR

---

*MinMax\_TR KEEL Preprocess Algorithm*


---

**Description**

MinMax\_TR Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test

```

MLP\_BP\_C

*MLP\_BP\_C KEEL Classification Algorithm***Description**

MLP\_BP\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
hidden_layers	hidden_layers. Default value = 2
hidden_nodes	hidden_nodes. Default value = 15
transfer	transfer. Default value = "Htan"
eta	eta. Default value = 0.15
alpha	alpha. Default value = 0.1
lambda	lambda. Default value = 0.0
test_data	test_data. Default value = TRUE
validation_data	validation_data. Default value = FALSE
cross_validation	cross_validation. Default value = FALSE
cycles	cycles. Default value = 10000
improve	improve. Default value = 0.01
tipify_inputs	tipify_inputs. Default value = TRUE
save_all	save_all. Default value = FALSE
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

Examples

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

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

---

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

```

---

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
```

---

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

---

**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	<i>PDFC_C KEEL Classification Algorithm</i>
--------	---

---

**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	<i>PFKNN_C KEEL Classification Algorithm</i>
---------	--

---

**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	<i>PNN_C KEEL Classification Algorithm</i>
-------	--

---

**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.

---

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

---

**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	<i>Proportional_D KEEL Preprocess Algorithm</i>
----------------	---

---

**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_itations_without_converge. Default value = 100

enviromentSize	enviromentSize. Default value = 3
numParticles	numParticles. Default value = 100
x	x. Default value = 0.72984
c1	c1. Default value = 2.05
c2	c2. Default value = 2.05
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

PSRCG\_TSS

---

*PSRCG\_TSS KEEL Preprocess Algorithm*


---

**Description**

PSRCG\_TSS Preprocess Algorithm from KEEL.

**Usage**

```
PSRCG_TSS(train, test, distance)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
distance	distance. Default value = "Euclidean"

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test

```

PUBLIC\_C

*PUBLIC\_C KEEL Classification Algorithm***Description**

PUBLIC\_C Classification Algorithm from KEEL.

**Usage**

```
PUBLIC_C(train, test, nodesBetweenPrune, estimateToPrune)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
nodesBetweenPrune	nodesBetweenPrune. Default value = 25
estimateToPrune	estimateToPrune. Default value = "PUBLIC(1)"

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

PW\_C

*PW\_C KEEL Classification Algorithm***Description**

PW\_C Classification Algorithm from KEEL.

**Usage**

```
PW_C(train, test, beta, ro, epsilon)
```

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

QDA\_C

*QDA\_C KEEL Classification Algorithm***Description**

QDA\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

RBFN\_C

*RBFN\_C KEEL Classification Algorithm***Description**

RBFN\_C Classification Algorithm from KEEL.

**Usage**

```
RBFN_C(train, test, neurons, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
neurons	neurons. Default value = 50
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```

---

RBFN\_R

*RBFN\_R KEEL Regression Algorithm*


---

**Description**

RBFN\_R Regression Algorithm from KEEL.

**Usage**

```
RBFN_R(train, test, neurons, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
neurons	neurons. Default value = 50
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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



**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions

```

---

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

---

**Description**

Method for read datasets in .dat KEEL format

**Usage**

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

**Arguments**

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

**Value**

Returns a data.frame object with the dataset

---

RegressionAlgorithm	<i>Regression Algorithm</i>
---------------------	-----------------------------

---

**Description**

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

---

RegressionResults	<i>Regression Results</i>
-------------------	---------------------------

---

**Description**

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

---

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

---

**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	<i>RISE_C KEEL Classification Algorithm</i>
--------	---

---

**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
```

---

runParallel	<i>Run Parallel</i>
-------------	---------------------

---

**Description**

Run a set of RKEEL algorithms in parallel

**Usage**

runParallel(algorithmList, cores)

**Arguments**

`algorithmList` List of RKEEL Algorithms to be executed

`cores` Number of cores to execute in parallel. If it is not specified, it detects the cores automatically and execute the experiment in all of them

**Value**

Returns a list with the executed algorithms

**Examples**

```
#Load datasets
iris_train <- RKEEL::loadKeelDataset("iris_train")
iris_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithms
learner_C45_C <- RKEEL::C45_C(iris_train, iris_test)
learner_FRNN_C <- RKEEL::FRNN_C(iris_train, iris_test)
learner_FuzzyKNN_C <- RKEEL::FuzzyKNN_C(iris_train, iris_test)
learner_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)

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

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

---

runSequential

*Run Sequential*


---

**Description**

Run a set of RKEEL algorithms in sequential.

**Usage**

```
runSequential(algorithmList)
```

**Arguments**

`algorithmList` List of RKEEL Algorithms to be executed

**Value**

Returns a list with the executed algorithms

**Examples**

```
#Load datasets
iris_train <- RKEEL::loadKeelDataset("iris_train")
iris_test <- RKEEL::loadKeelDataset("iris_test")

#Create algorithms
learner_C45_C <- RKEEL::C45_C(iris_train, iris_test)
learner_FRNN_C <- RKEEL::FRNN_C(iris_train, iris_test)
learner_FuzzyKNN_C <- RKEEL::FuzzyKNN_C(iris_train, iris_test)
learner_KNN_C <- RKEEL::KNN_C(iris_train, iris_test)
learner_Logistic_C <- RKEEL::Logistic_C(iris_train, iris_test)
learner_LDA_C <- RKEEL::LDA_C(iris_train, iris_test)

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

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

---

SaturationFilter\_F      *SaturationFilter\_F KEEL Preprocess Algorithm*

---

**Description**

SaturationFilter\_F Preprocess Algorithm from KEEL.

**Usage**

```
SaturationFilter_F(train, test, seed)
```

**Arguments**

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

**Value**

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

**Examples**

```

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

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

#Run algorithm
#algorithm$run()

#See results
#algorithm$preprocessed_test

```

SFS\_IEP\_FS

*SFS\_IEP\_FS KEEL Preprocess Algorithm***Description**

SFS\_IEP\_FS Preprocess Algorithm from KEEL.

**Usage**

```
SFS_IEP_FS(train, test, threshold, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
threshold	threshold. Default value = 0.005
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

```

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test

```

SGA\_C

*SGA\_C KEEL Classification Algorithm***Description**

SGA\_C Classification Algorithm from KEEL.

**Usage**

```
SGA_C(train, test, mut_prob_1to0, mut_prob_0to1, cross_prob,
       pop_size, evaluations, alfa, selection_type, k,
       distance, seed)
```

**Arguments**

train	Train dataset as a data.frame object
test	Test dataset as a data.frame object
mut_prob_1to0	mut_prob_1to0. Default value = 0.01
mut_prob_0to1	mut_prob_0to1. Default value = 0.001
cross_prob	cross_prob. Default value = 1
pop_size	pop_size. Default value = 50
evaluations	evaluations. Default value = 10000
alfa	alfa. Default value = 0.5
selection_type	selection_type. Default value = "orden_based"
k	k. Default value = 1
distance	distance. Default value = "Euclidean"
seed	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$testPredictions
```



---

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

---

**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	<i>Slipper_C KEEL Classification Algorithm</i>
-----------	--

---

**Description**

Slipper\_C Classification Algorithm from KEEL.

**Usage**

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

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>grow_pct</code>	grow_pct. Default value = 0.66
<code>numBoosting</code>	numBoosting. Default value = 100
<code>seed</code>	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

A data.frame with the 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**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>C</code>	C. Default value = 1.0
<code>toleranceParameter</code>	toleranceParameter. Default value = 0.001
<code>epsilon</code>	epsilon. Default value = 1.0e-12
<code>RBFKernel_gamma</code>	RBFKernel_gamma. Default value = 0.01
<code>normalized_PolyKernel_exponent</code>	normalized_PolyKernel_exponent. Default value = 1
<code>normalized_PolyKernel_useLowerOrder</code>	normalized_PolyKernel_useLowerOrder. Default value = FALSE
<code>PukKernel_omega</code>	PukKernel_omega. Default value = 1.0
<code>PukKernel_sigma</code>	PukKernel_sigma. Default value = 1.0
<code>StringKernel_lambda</code>	StringKernel_lambda. Default value = 0.5
<code>StringKernel_subsequenceLength</code>	StringKernel_subsequenceLength. Default value = 3
<code>StringKernel_maxSubsequenceLength</code>	StringKernel_maxSubsequenceLength. Default value = 9
<code>StringKernel_normalize</code>	StringKernel_normalize. Default value = FALSE
<code>StringKernel_pruning</code>	StringKernel_pruning. Default value = "None"
<code>KernelType</code>	KernelType. Default value = "PolyKernel"
<code>FitLogisticModels</code>	FitLogisticModels. Default value = FALSE
<code>ConvertNominalAttributesToBinary</code>	ConvertNominalAttributesToBinary. Default value = TRUE
<code>PreprocessType</code>	PreprocessType. Default value = "Normalize"
<code>seed</code>	Seed for random numbers. If it is not assigned a value, the seed will be a random number

**Value**

A data.frame with the actual and predicted 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	<i>Thrift_R KEEL Regression Algorithm</i>
----------	---

---

**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**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object
<code>numlabels</code>	numlabels. Default value = 5
<code>KB</code>	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**

<code>data</code>	data.frame dataset
<code>fileName</code>	String with the file name to store the dataset

**Examples**

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

---

`writeDatFromDataframes`*Write .dat from data.frames*

---

**Description**

Method for writing both train and test .dat dataset files in KEEL format.

**Usage**

```
writeDatFromDataframes(trainData, testData,  
                        trainFileName, testFileName)
```

**Arguments**

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

---

`ZScore_TR`*ZScore\_TR KEEL Preprocess Algorithm*

---

**Description**

ZScore\_TR Preprocess Algorithm from KEEL.

**Usage**

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

**Arguments**

<code>train</code>	Train dataset as a data.frame object
<code>test</code>	Test dataset as a data.frame object

**Value**

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

**Examples**

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

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

#Run algorithm
algorithm$run()

#See results
algorithm$preprocessed_test
```

# Index

## \*Topic **classification**

AdaBoostNC\_C, [5](#)  
ART\_C, [8](#)  
AssociativeClassificationAlgorithm,  
[9](#)  
BNGE\_C, [10](#)  
Bojarczuk\_GP\_C, [11](#)  
BSE\_C, [12](#)  
C45\_C, [14](#)  
C45Binarization\_C, [12](#)  
C45Rules\_C, [13](#)  
C\_SVM\_C, [26](#)  
CamNN\_C, [15](#)  
CART\_C, [16](#)  
CenterNN\_C, [17](#)  
CFAR\_C, [18](#)  
CFKNN\_C, [19](#)  
CHC\_C, [20](#)  
ClassificationAlgorithm, [21](#)  
ClassificationResults, [21](#)  
CNN\_C, [23](#)  
CPW\_C, [24](#)  
CW\_C, [25](#)  
DecrRBFN\_C, [27](#)  
Deeps\_C, [28](#)  
DSM\_C, [29](#)  
DT\_GA\_C, [30](#)  
Falco\_GP\_C, [32](#)  
FCRA\_C, [33](#)  
FRNN\_C, [34](#)  
FURIA\_C, [36](#)  
FuzzyFARCHD\_C, [37](#)  
FuzzyKNN\_C, [38](#)  
FuzzyNPC\_C, [39](#)  
GANN\_C, [39](#)  
GFS\_AdaBoost\_C, [42](#)  
GFS\_LogitBoost\_C, [45](#)  
ID3\_C, [48](#)  
IF\_KNN\_C, [50](#)

IncrRBFN\_C, [51](#)  
JFKNN\_C, [54](#)  
Kernel\_C, [55](#)  
KNN\_C, [56](#)  
KSNN\_C, [58](#)  
KStar\_C, [59](#)  
LDA\_C, [60](#)  
LinearLMS\_C, [60](#)  
Logistic\_C, [63](#)  
MLP\_BP\_C, [67](#)  
NB\_C, [71](#)  
NM\_C, [71](#)  
NNEP\_C, [72](#)  
NU\_SVM\_C, [74](#)  
PART\_C, [76](#)  
PDFC\_C, [76](#)  
PFKNN\_C, [78](#)  
PNN\_C, [78](#)  
PolQuadraticLMS\_C, [79](#)  
PRISM\_C, [82](#)  
PSO\_ACO\_C, [83](#)  
PUBLIC\_C, [85](#)  
PW\_C, [86](#)  
QDA\_C, [87](#)  
RBFN\_C, [87](#)  
Ripper\_C, [91](#)  
RISE\_C, [92](#)  
SGA\_C, [96](#)  
Shrink\_C, [97](#)  
Slipper\_C, [97](#)  
SMO\_C, [98](#)  
Tan\_GP\_C, [101](#)  
VWFuzzyKNN\_C, [105](#)

## \*Topic **preprocess**

ABB\_IEP\_FS, [4](#)  
AllKNN\_TSS, [6](#)  
AllPosible\_MV, [7](#)  
ANR\_F, [8](#)  
Bayesian\_D, [9](#)

- CleanAttributes\_TR, 21
- ClusterAnalysis\_D, 22
- DecimalScaling\_TR, 27
- ID3\_D, 49
- Ignore\_MV, 51
- IterativePartitioningFilter\_F, 53
- KMeans\_MV, 55
- KNN\_MV, 57
- LVF\_IEP\_FS, 63
- MinMax\_TR, 66
- ModelCS\_TSS, 69
- MostCommon\_MV, 70
- Nominal2Binary\_TR, 73
- POP\_TSS, 81
- PreprocessAlgorithm, 81
- Proportional\_D, 82
- PSRCG\_TSS, 84
- Relief\_FS, 90
- SaturationFilter\_F, 94
- SFS\_IEP\_FS, 95
- SSGA\_Integer\_knn\_FS, 100
- UniformFrequency\_D, 103
- UniformWidth\_D, 104
- ZScore\_TR, 107
- \*Topic **regression**
  - CART\_R, 17
  - EPSILON\_SVR\_R, 31
  - FRSBM\_R, 35
  - GFS\_GP\_R, 43
  - GFS\_GSP\_R, 44
  - GFS\_RB\_MF\_R, 46
  - LinearLMS\_R, 61
  - M5\_R, 65
  - M5Rules\_R, 64
  - MLP\_BP\_R, 68
  - NU\_SVR\_R, 75
  - PolQuadraticLMS\_R, 80
  - RBFN\_R, 88
  - RegressionAlgorithm, 89
  - RegressionResults, 90
  - Thrift\_R, 102
  - WM\_R, 105
- \*Topic **utils**
  - getAttributeLinesFromDataframes, 41
  - hasContinuousData, 47
  - hasMissingValues, 48
  - isMultiClass, 52
  - loadKeelDataset, 62
  - read.keel, 89
  - runParallel, 92
  - runSequential, 93
  - writeDatFromDataframe, 106
  - writeDatFromDataframes, 107
- ABB\_IEP\_FS, 4
- AdaBoostNC\_C, 5
- AllKNN\_TSS, 6
- AllPossible\_MV, 7
- ANR\_F, 8
- ART\_C, 8
- AssociativeClassificationAlgorithm, 9
- Bayesian\_D, 9
- BNGE\_C, 10
- Bojarczuk\_GP\_C, 11
- BSE\_C, 12
- C45\_C, 14
- C45Binarization\_C, 12
- C45Rules\_C, 13
- C\_SVM\_C, 26
- CamNN\_C, 15
- CART\_C, 16
- CART\_R, 17
- CenterNN\_C, 17
- CFAR\_C, 18
- CFKNN\_C, 19
- CHC\_C, 20
- ClassificationAlgorithm, 21
- ClassificationResults, 21
- CleanAttributes\_TR, 21
- ClusterAnalysis\_D, 22
- CNN\_C, 23
- CPW\_C, 24
- CW\_C, 25
- DecimalScaling\_TR, 27
- DecrRBFN\_C, 27
- Deeps\_C, 28
- DSM\_C, 29
- DT\_GA\_C, 30
- EPSILON\_SVR\_R, 31
- Falco\_GP\_C, 32
- FCRA\_C, 33
- FRNN\_C, 34

- FRSBM\_R, [35](#)
- FURIA\_C, [36](#)
- FuzzyFARCHD\_C, [37](#)
- FuzzyKNN\_C, [38](#)
- FuzzyNPC\_C, [39](#)
  
- GANN\_C, [39](#)
- getAttributeLinesFromDataframes, [41](#)
- GFS\_AdaBoost\_C, [42](#)
- GFS\_GP\_R, [43](#)
- GFS\_GSP\_R, [44](#)
- GFS\_LogitBoost\_C, [45](#)
- GFS\_RB\_MF\_R, [46](#)
  
- hasContinuousData, [47](#)
- hasMissingValues, [48](#)
  
- ID3\_C, [48](#)
- ID3\_D, [49](#)
- IF\_KNN\_C, [50](#)
- Ignore\_MV, [51](#)
- IncrRBFN\_C, [51](#)
- isMultiClass, [52](#)
- IterativePartitioningFilter\_F, [53](#)
  
- JFKNN\_C, [54](#)
  
- KeelAlgorithm, [54](#)
- Kernel\_C, [55](#)
- KMeans\_MV, [55](#)
- KNN\_C, [56](#)
- KNN\_MV, [57](#)
- KSNN\_C, [58](#)
- KStar\_C, [59](#)
  
- LDA\_C, [60](#)
- LinearLMS\_C, [60](#)
- LinearLMS\_R, [61](#)
- loadKeelDataset, [62](#)
- Logistic\_C, [63](#)
- LVF\_IEP\_FS, [63](#)
  
- M5\_R, [65](#)
- M5Rules\_R, [64](#)
- MinMax\_TR, [66](#)
- MLP\_BP\_C, [67](#)
- MLP\_BP\_R, [68](#)
- ModelCS\_TSS, [69](#)
- MostCommon\_MV, [70](#)
  
- NB\_C, [71](#)
- NM\_C, [71](#)
- NNEP\_C, [72](#)
- Nominal2Binary\_TR, [73](#)
- NU\_SVM\_C, [74](#)
- NU\_SVR\_R, [75](#)
  
- PART\_C, [76](#)
- PDFC\_C, [76](#)
- PFKNN\_C, [78](#)
- PNN\_C, [78](#)
- PolQuadraticLMS\_C, [79](#)
- PolQuadraticLMS\_R, [80](#)
- POP\_TSS, [81](#)
- PreprocessAlgorithm, [81](#)
- PRISM\_C, [82](#)
- Proportional\_D, [82](#)
- PSO\_ACO\_C, [83](#)
- PSRCG\_TSS, [84](#)
- PUBLIC\_C, [85](#)
- PW\_C, [86](#)
  
- QDA\_C, [87](#)
  
- R6\_ABB\_IEP\_FS (ABB\_IEP\_FS), [4](#)
- R6\_AdaBoostNC\_C (AdaBoostNC\_C), [5](#)
- R6\_AllKNN\_TSS (AllKNN\_TSS), [6](#)
- R6\_AllPosible\_MV (AllPosible\_MV), [7](#)
- R6\_ANR\_F (ANR\_F), [8](#)
- R6\_ART\_C (ART\_C), [8](#)
- R6\_Bayesian\_D (Bayesian\_D), [9](#)
- R6\_BNGE\_C (BNGE\_C), [10](#)
- R6\_Bojarczuk\_GP\_C (Bojarczuk\_GP\_C), [11](#)
- R6\_BSE\_C (BSE\_C), [12](#)
- R6\_C45\_C (C45\_C), [14](#)
- R6\_C45Binarization\_C  
(C45Binarization\_C), [12](#)
- R6\_C45Rules\_C (C45Rules\_C), [13](#)
- R6\_C\_SVM\_C (C\_SVM\_C), [26](#)
- R6\_CamNN\_C (CamNN\_C), [15](#)
- R6\_CART\_C (CART\_C), [16](#)
- R6\_CART\_R (CART\_R), [17](#)
- R6\_CenterNN\_C (CenterNN\_C), [17](#)
- R6\_CFAR\_C (CFAR\_C), [18](#)
- R6\_CFKNN\_C (CFKNN\_C), [19](#)
- R6\_CHC\_C (CHC\_C), [20](#)
- R6\_CleanAttributes\_TR  
(CleanAttributes\_TR), [21](#)

- R6\_ClusterAnalysis\_D  
(ClusterAnalysis\_D), 22
- R6\_CNN\_C (CNN\_C), 23
- R6\_CPW\_C (CPW\_C), 24
- R6\_CW\_C (CW\_C), 25
- R6\_DecimalScaling\_TR  
(DecimalScaling\_TR), 27
- R6\_DecrRBFN\_C (DecrRBFN\_C), 27
- R6\_Deeps\_C (Deeps\_C), 28
- R6\_DSM\_C (DSM\_C), 29
- R6\_DT\_GA\_C (DT\_GA\_C), 30
- R6\_EPSILON\_SVR\_R (EPSILON\_SVR\_R), 31
- R6\_Falco\_GP\_C (Falco\_GP\_C), 32
- R6\_FCRA\_C (FCRA\_C), 33
- R6\_FRNN\_C (FRNN\_C), 34
- R6\_FRSBM\_R (FRSBM\_R), 35
- R6\_FURIA\_C (FURIA\_C), 36
- R6\_FuzzyFARCHD\_C (FuzzyFARCHD\_C), 37
- R6\_FuzzyKNN\_C (FuzzyKNN\_C), 38
- R6\_FuzzyNPC\_C (FuzzyNPC\_C), 39
- R6\_GANN\_C (GANN\_C), 39
- R6\_GFS\_AdaBoost\_C (GFS\_AdaBoost\_C), 42
- R6\_GFS\_GP\_R (GFS\_GP\_R), 43
- R6\_GFS\_GSP\_R (GFS\_GSP\_R), 44
- R6\_GFS\_LogitBoost\_C (GFS\_LogitBoost\_C),  
45
- R6\_GFS\_RB\_MF\_R (GFS\_RB\_MF\_R), 46
- R6\_ID3\_C (ID3\_C), 48
- R6\_ID3\_D (ID3\_D), 49
- R6\_IF\_KNN\_C (IF\_KNN\_C), 50
- R6\_Ignore\_MV (Ignore\_MV), 51
- R6\_IncrRBFN\_C (IncrRBFN\_C), 51
- R6\_IterativePartitioningFilter\_F  
(IterativePartitioningFilter\_F),  
53
- R6\_JFKNN\_C (JFKNN\_C), 54
- R6\_Kernel\_C (Kernel\_C), 55
- R6\_KMeans\_MV (KMeans\_MV), 55
- R6\_KNN\_C (KNN\_C), 56
- R6\_KNN\_MV (KNN\_MV), 57
- R6\_KSNN\_C (KSNN\_C), 58
- R6\_KStar\_C (KStar\_C), 59
- R6\_LDA\_C (LDA\_C), 60
- R6\_LinearLMS\_C (LinearLMS\_C), 60
- R6\_LinearLMS\_R (LinearLMS\_R), 61
- R6\_Logistic\_C (Logistic\_C), 63
- R6\_LVF\_IEP\_FS (LVF\_IEP\_FS), 63
- R6\_M5\_R (M5\_R), 65
- R6\_M5Rules\_R (M5Rules\_R), 64
- R6\_MinMax\_TR (MinMax\_TR), 66
- R6\_MLP\_BP\_C (MLP\_BP\_C), 67
- R6\_MLP\_BP\_R (MLP\_BP\_R), 68
- R6\_ModelCS\_TSS (ModelCS\_TSS), 69
- R6\_MostCommon\_MV (MostCommon\_MV), 70
- R6\_NB\_C (NB\_C), 71
- R6\_NM\_C (NM\_C), 71
- R6\_NNEP\_C (NNEP\_C), 72
- R6\_Nominal2Binary\_TR  
(Nominal2Binary\_TR), 73
- R6\_NU\_SVM\_C (NU\_SVM\_C), 74
- R6\_NU\_SVR\_R (NU\_SVR\_R), 75
- R6\_PART\_C (PART\_C), 76
- R6\_PDFC\_C (PDFC\_C), 76
- R6\_PFKNN\_C (PFKNN\_C), 78
- R6\_PNN\_C (PNN\_C), 78
- R6\_PolQuadraticLMS\_C  
(PolQuadraticLMS\_C), 79
- R6\_PolQuadraticLMS\_R  
(PolQuadraticLMS\_R), 80
- R6\_POP\_TSS (POP\_TSS), 81
- R6\_PRISM\_C (PRISM\_C), 82
- R6\_Proportional\_D (Proportional\_D), 82
- R6\_PSO\_ACO\_C (PSO\_ACO\_C), 83
- R6\_PSRCG\_TSS (PSRCG\_TSS), 84
- R6\_PUBLIC\_C (PUBLIC\_C), 85
- R6\_PW\_C (PW\_C), 86
- R6\_QDA\_C (QDA\_C), 87
- R6\_RBFN\_C (RBFN\_C), 87
- R6\_RBFN\_R (RBFN\_R), 88
- R6\_Relief\_FS (Relief\_FS), 90
- R6\_Ripper\_C (Ripper\_C), 91
- R6\_RISE\_C (RISE\_C), 92
- R6\_SaturationFilter\_F  
(SaturationFilter\_F), 94
- R6\_SFS\_IEP\_FS (SFS\_IEP\_FS), 95
- R6\_SGA\_C (SGA\_C), 96
- R6\_Shrink\_C (Shrink\_C), 97
- R6\_Slipper\_C (Slipper\_C), 97
- R6\_SMO\_C (SMO\_C), 98
- R6\_SSGA\_Integer\_knn\_FS  
(SSGA\_Integer\_knn\_FS), 100
- R6\_Tan\_GP\_C (Tan\_GP\_C), 101
- R6\_Thrift\_R (Thrift\_R), 102
- R6\_UniformFrequency\_D  
(UniformFrequency\_D), 103
- R6\_UniformWidth\_D (UniformWidth\_D), 104



R6\_VWFuzzyKNN\_C (VWFuzzyKNN\_C), [105](#)  
R6\_WM\_R (WM\_R), [105](#)  
R6\_ZScore\_TR (ZScore\_TR), [107](#)  
RBFN\_C, [87](#)  
RBFN\_R, [88](#)  
read.keel, [89](#)  
RegressionAlgorithm, [89](#)  
RegressionResults, [90](#)  
Relief\_FS, [90](#)  
Ripper\_C, [91](#)  
RISE\_C, [92](#)  
runParallel, [92](#)  
runSequential, [93](#)  
  
SaturationFilter\_F, [94](#)  
SFS\_IEP\_FS, [95](#)  
SGA\_C, [96](#)  
Shrink\_C, [97](#)  
Slipper\_C, [97](#)  
SMO\_C, [98](#)  
SSGA\_Integer\_knn\_FS, [100](#)  
  
Tan\_GP\_C, [101](#)  
Thrift\_R, [102](#)  
  
UniformFrequency\_D, [103](#)  
UniformWidth\_D, [104](#)  
  
VWFuzzyKNN\_C, [105](#)  
  
WM\_R, [105](#)  
writeDatFromDataframe, [106](#)  
writeDatFromDataframes, [107](#)  
  
ZScore\_TR, [107](#)