Package ‘utiml’

July 31, 2017

Type Package

Title Utilities for Multi-Label Learning

Version 0.1.3

Date 2017-07-28


URL https://github.com/rivolli/utiml

Depends R (>= 3.0.0), mldr(>= 0.3.22)

Imports stats, utils

Suggests C50, e1071, FSelector, infotheo, kknn, knitr, parallel, randomForest, rJava(>= 0.9), rmarkdown, rpart, RWeka(>= 0.4), testthat, xgboost(>= 0.6-4)

License GPL | file LICENSE

LazyData true

BugReports https://github.com/rivolli/utiml

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Adriano Rivolli [aut, cre]

Maintainer Adriano Rivolli <rivolli@utfpr.edu.br>

Repository CRAN

Date/Publication 2017-07-31 21:57:08 UTC
R topics documented:

+.mlconfmat ................................................. 5
as.bipartition ............................................. 6
as.matrix.mlresult ......................................... 6
as.mlresult .................................................. 7
as.probability .............................................. 8
as.ranking ................................................... 8
baseline ...................................................... 9
br ............................................................ 10
brplus ......................................................... 11
cc ........................................................... 12
clr .......................................................... 14
compute_multilabel_predictions .......................... 15
create_holdout_partition ................................... 16
create_kfold_partition ..................................... 18
create_random_subset ...................................... 19
create_subset ............................................... 20
ctrl ........................................................ 21
dbr ........................................................ 23
ebr ........................................................ 24
ecc ........................................................ 26
eps .......................................................... 27
fill_sparse_mldata .......................................... 29
fixed_threshold ............................................. 29
homer ......................................................... 31
is.bipartition .............................................. 32
is.probability ............................................... 33
lcard_threshold ............................................. 33
lift .......................................................... 34
lp ........................................................... 35
mbr ........................................................ 36
mcut_threshold ............................................. 38
merge_mlconfmat ........................................... 39
mldata ....................................................... 40
mlpredict ..................................................... 40
mltrain ....................................................... 43
multilabel_confusion_matrix ............................... 45
multilabel_evaluate ........................................ 47
multilabel_measures ....................................... 48
multilabel_prediction ..................................... 49
normalize_mldata .......................................... 50
ns ........................................................... 50
partition_fold .............................................. 52
pcut_threshold ............................................. 53
ppt .......................................................... 54
predict.BASELINEmodel ................................... 55
predict.BRmodel ........................................... 56
R topics documented:

predict.BRPmodel .......................................................... 57
predict.CCmodel .......................................................... 59
predict.CLRmodel .......................................................... 60
predict.CTRLmodel ......................................................... 61
predict.DBRmodel .......................................................... 62
predict.EBRmodel .......................................................... 63
predict.ECCmodel .......................................................... 64
predict.EPSmodel .......................................................... 66
predict.HOMERmodel ....................................................... 67
predict.LIFTmodel .......................................................... 68
predict.LPmodel ........................................................... 69
predict.MBRmodel .......................................................... 70
predict.NSmodel ........................................................... 71
predict.PPTmodel .......................................................... 72
predict.PruDentmodel ....................................................... 73
predict.PSmodel ........................................................... 74
predict.RAkELmodel ....................................................... 75
predict.RDBRmodel ......................................................... 76
predict.RPCmodel .......................................................... 77
print.BRmodel .............................................................. 78
print.BRPmodel ............................................................. 79
print.CCmodel .............................................................. 79
print.CLRmodel .............................................................. 80
print.CTRLmodel ........................................................... 80
print.DBRmodel ............................................................. 81
print.EBRmodel ............................................................. 81
print.ECCmodel ............................................................. 82
print.EPSmodel ............................................................. 82
print.kFoldPartition ......................................................... 83
print.LIFTmodel ............................................................. 83
print.LPmodel ............................................................... 84
print.majorityModel ......................................................... 84
print.MBRmodel ............................................................. 85
print.mlconfmat ............................................................ 85
print.mlresult .............................................................. 86
print.NSmodel ............................................................... 86
print.PPTmodel .............................................................. 87
print.PruDentmodel ........................................................ 87
print.PSmodel ............................................................... 88
print.RAkELmodel ........................................................ 88
print.randomModel ......................................................... 89
print.RDBRmodel .......................................................... 89
print.RPCmodel ............................................................ 90
prudent ................................................................. 90
ps ................................................................. 91
rakel ................................................................. 93
rcut_threshold ............................................................ 94
rdbr ................................................................. 95
remove_attributes .......................................................... 97
remove_labels ................................................................. 97
remove_skewness_labels ..................................................... 98
remove_unique_attributes ................................................... 99
remove_unlabeled_instances ............................................... 100
replace_nominal_attributes ................................................. 100
rpc .............................................................. 101
scut_threshold ............................................................... 102
subset_correction ........................................................... 104
summary.mltransformation ............................................... 105
toyml ............................................................... 105
utiml ............................................................ 107
utiml_all_measures_names ............................................... 108
utiml_compute_ensemble .................................................. 108
utiml_ensemble_average .................................................. 109
utiml_ensemble_check_votschema ....................................... 109
utiml_ensemble_majority .................................................. 110
utiml_ensemble_maximum .................................................. 110
utiml_ensemble_method .................................................... 111
utiml_ensemble_minimum ................................................... 111
utiml_ifelse ............................................................ 112
utiml_is_equal_sets ........................................................ 112
utiml_iterative_split ....................................................... 113
utiml_labels_correlation .................................................. 114
utiml_labels_IG ........................................................... 115
utiml_lapply ............................................................. 115
utiml_measure_accuracy .................................................... 116
utiml_measure_average_precision ...................................... 116
utiml_measure_binary_accuracy .......................................... 117
utiml_measure_binary_AUC ................................................ 117
utiml_measure_binary_balace .............................................. 118
utiml_measure_binary_f1 .................................................... 118
utiml_measure_binary_precision ........................................ 119
utiml_measure_binary_recall .............................................. 119
utiml_measure_coverage ................................................... 120
utiml_measure_f1 .......................................................... 120
utiml_measure_hamming_loss .............................................. 121
utiml_measure_is_error ..................................................... 121
utiml_measure_macro_accuracy .......................................... 122
utiml_measure_macro_AUC ................................................ 122
utiml_measure_macro_f1 .................................................... 123
utiml_measure_macro_precision .......................................... 123
utiml_measure_macro_recall .............................................. 124
utiml_measure_margin_loss ................................................. 124
utiml_measure_micro_accuracy ............................................ 125
utiml_measure_micro_AUC .................................................. 125
utiml_measure_micro_f1 .................................................... 126
utiml_measure_micro_precision .......................................... 126
+.mlconfmat

Join two multi-label confusion matrix

Description

Join two multi-label confusion matrix

Usage

```r
## S3 method for class 'mlconfmat'
mlcm1 + mlcm2
```

Arguments

- `mlcm1`: A `mlconfmat`
- `mlcm2`: Other `mlconfmat`

Value

`mlconfmat`
### as.bipartition

*Convert a mlresult to a bipartition matrix*

**Description**

Convert a mlresult to a bipartition matrix

**Usage**

```r
as.bipartition(mlresult)
```

**Arguments**

- `mlresult`: The mlresult object

**Value**

matrix with bipartition values

### as.matrix.mlresult

*Convert a mlresult to matrix*

**Description**

Convert a mlresult to matrix

**Usage**

```r
## S3 method for class 'mlresult'
as.matrix(x, ...)
```

**Arguments**

- `x`: The mlresult object
- `...`: ignored

**Value**

matrix
as.mlresult

Convert a matrix prediction in a multi label prediction

Description

Convert a matrix prediction in a multi label prediction

Usage

as.mlresult(predictions, probability = TRUE, ...)

## Default S3 method:
as.mlresult(predictions, probability = TRUE, ...
threshold = 0.5)

## S3 method for class 'mlresult'
as.mlresult(predictions, probability = TRUE, ...)

Arguments

predictions a Matrix or data.frame contained the scores/probabilities values. The columns are the labels and the rows are the examples.
probability A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: TRUE)
... ignored
threshold A single value between 0 and 1 or a list with threshold values contained one value per label (Default: 0.5). Only used when the predictions are not a mlresult.

Value

An object of type mlresult

Methods (by class)

• default: Default mlresult transform method
• mlresult: change the mlresult type

Examples

predictions <- matrix(runif(100), ncol = 10)
colnames(predictions) <- paste('label', 1:10, sep='')

# Create a mlresult from a matrix
mlresult <- as.mlresult(predictions)
mlresult <- as.mlresult(predictions, probability = FALSE)
mlresult <- as.mlresult(predictions, probability = FALSE, threshold = 0.6)
as.probability  
Convert a mlresult to a probability matrix

Description
Convert a mlresult to a probability matrix

Usage
as.probability(mlresult)

Arguments
mlresult  
The mlresult object

Value
matrix with probabilities values

as.ranking  
Convert a mlresult to a ranking matrix

Description
Convert a mlresult to a ranking matrix

Usage
as.ranking(mlresult, ties.method = "min", ...)

Arguments
mlresult  
The mlresult object
ties.method  
A character string specifying how ties are treated (Default: "min"). see rank to more details.
...  
Others parameters passed to the rank method.

Value
matrix with ranking values
Description

Create a baseline model for multilabel classification.

Usage

```
baseline(mdata, metric = c("general", "F1", "hamming-loss", "subset-accuracy"), ...)
```

Arguments

- `mdata` A mldr dataset used to train the binary models.
- `metric` Define the strategy used to predict the labels.
  
  The possible values are: 'general', 'F1', 'hamming-loss' or 'subset-accuracy'.
  
  See the description for more details. (Default: 'general').

Details

Baseline is a naive multi-label classifier that maximize/minimize a specific measure without induces a learning model. It uses the general information about the labels in training dataset to estimate the labels in a test dataset.

The follow strategies are available:

- **general** Predict the k most frequent labels, where k is the integer most close of label cardinality.
- **F1** Predict the most frequent labels that obtain the best F1 measure in training data. In the original paper, the authors use the less frequent labels.
- **hamming-loss** Predict the labels that are associated with more than 50% of instances.
- **subset-accuracy** Predict the most common labelset.

Value

An object of class `baselinemodel` containing the set of fitted models, including:

- `labels` A vector with the label names.
- `predict` A list with the labels that will be predicted.

References

Examples

```r
model <- baseline(toyml)
pred <- predict(model, toyml)

## Change the metric
model <- baseline(toyml, "F1")
model <- baseline(toyml, "subset-accuracy")
```

---

**Description**

Create a Binary Relevance model for multilabel classification.

**Usage**

```r
br(mdata, base.algorithm = getOption("utilm.base.algorithm", "SVM"), ...,
cores = getOption("utilm.cores", 1), seed = getOption("utilm.seed", NA))
```

**Arguments**

- `mdata` A mlr dataset used to train the binary models.
- `base.algorithm` A string with the name of the base algorithm (Default: `options("utilm.base.algorithm", "SVM")`)
- `...` Others arguments passed to the base algorithm for all subproblems
- `cores` The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `options("utilm.cores", 1)`)  
- `seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `options("utilm.seed", NA)`)  

**Details**

Binary Relevance is a simple and effective transformation method to predict multi-label data. This is based on the one-versus-all approach to build a specific model for each label.

**Value**

An object of class `bRmodel` containing the set of fitted models, including:

- `labels` A vector with the label names.
- `models` A list of the generated models, named by the label names.

**References**

See Also

Other Transformation methods: brplus, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc

Examples

```r
model <- br(toym1, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use SVM as base algorithm
model <- br(toym1, "SVM")
pred <- predict(model, toyml)

# Change the base algorithm and use 4 CORES
model <- br(toym1[1:50], 'RF', cores = 4, seed = 123)

# Set a parameters for all subproblems
model <- br(toym1, 'KNN', k=5)

## End(Not run)
```

---

**brplus**

**BR+ or BRplus for multi-label Classification**

**Description**

Create a BR+ classifier to predict multi-label data. This is a simple approach that enables the binary classifiers to discover existing label dependency by themselves. The main idea of BR+ is to increment the feature space of the binary classifiers to let them discover existing label dependency by themselves.

**Usage**

```r
brplus(mdata, base.algorithm =getOption("utiml.base.algorith", "SVM"), ..., cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```

**Arguments**

- `mdata`: A mlr dataset used to train the binary models.
- `base.algorithm`: A string with the name of the base algorithm. (Default: options("utiml.base.algorith", "SVM"))
- `...`: Others arguments passed to the base algorithm for all subproblems.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))
Details
This implementation has different strategy to predict the final set of labels for unlabeled examples, as proposed in original paper.

Value
An object of class `brpmodel` containing the set of fitted models, including:

- `freq` The label frequencies to use with the 'Stat' strategy
- `initial` The BR model to predict the values for the labels to initial step
- `models` A list of final models named by the label names.

References

See Also
Other Transformation methods: `br`, `cc`, `clr`, `ctrl`, `dbr`, `ebr`, `ecc`, `eps`, `homer`, `lift`, `lp`, `mbr`, `ns`, `ppt`, `prudent`, `ps`, `rakel`, `rdb`, `rpc`
Other Stacking methods: `mbr`

Examples
```
# Use SVM as base algorithm
model <- brplus(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use Random Forest as base algorithm and 4 cores
model <- brplus(toyml, 'RF', cores = 4, seed = 123)

## End(Not run)
```

Description
Create a Classifier Chains model for multilabel classification.

Usage
```
cc(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
   chain = NA, ..., cores = getOption("utiml.cores", 1),
   seed = getOption("utiml.seed", NA))
```
Arguments

mdata A mlr dataset used to train the binary models.
base.algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM")
chain A vector with the label names to define the chain order. If empty the chain is the
default label sequence of the dataset. (Default: NA)
... Others arguments passed to the base algorithm for all subproblems.
cores The number of cores to parallelize the training. Values higher than 1 require the
parallel package. (Default: options("utiml.cores", 1))
seed An optional integer used to set the seed. This is useful when the method is run
in parallel. (Default: options("utiml.seed", NA))

Details

Classifier Chains is a Binary Relevance transformation method based to predict multi-label data.
This is based on the one-versus-all approach to build a specific model for each label. It is different
from BR method due the strategy of extended the attribute space with the 0/1 label relevances of all
previous classifiers, forming a classifier chain.

Value

An object of class ccmodel containing the set of fitted models, including:

chain A vector with the chain order.
labels A vector with the label names in expected order.
models A list of models named by the label names.

References

sification. Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer
Science, 5782, 254-269.

See Also

Other Transformation methods: brplus, br, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr,
ns, ppt, prudent, ps, rakel, rdbr, rpc

Examples

model <- cc(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use a specific chain with J48 classifier
mychain <- sample(rownames(toyml$labels))
model <- cc(toyml, 'J48', mychain)

# Set a specific parameter
model <- cc(toyml, 'KNN', k=5)

# Run with multiple-cores
model <- cc(toyml, 'RF', cores = 5, seed = 123)

## End(Not run)

clr

Calibrated Label Ranking (CLR) for multi-label Classification

Description

Create a CLR model for multilabel classification.

Usage

clr(mdata, base.algorithn = getOption("utiml.base.algorithn", "SVM"), ..., 
cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))

Arguments

mdata A mlr dataset used to train the binary models.
base.algorithn A string with the name of the base algorithm. (Default: options("utiml.base.algorithn", "SVM"))
... Others arguments passed to the base algorithm for all subproblems
cores The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
seed An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Details

CLR is an extension of label ranking that incorporates the calibrated scenario. The introduction of an artificial calibration label, separates the relevant from the irrelevant labels.

Value

An object of class RPCmodel containing the set of fitted models, including:

labels A vector with the label names.
rpcmodel A RPC model.
brmodel A BR model used to calibrated the labels.
References


See Also

Other Transformation methods: brplus, br, cc, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc
Other Pairwise methods: rpc

Examples

```r
model <- clr(toyml, "RANDOM")
pred <- predict(model, toyml)

# Not run:
```

calculate_multilabel_predictions

- **Compute the multi-label ensemble predictions based on some vote schema**

Description

Compute the multi-label ensemble predictions based on some vote schema

Usage

```r
compute_multilabel_predictions(predictions, vote.schema = "maj", probability = getOption("utiml.use.probs", TRUE))

utiml_predict_ensemble(predictions, vote.schema, probability)
```

Arguments

- **predictions** A list of multi-label predictions (mlresult).
- **vote.schema** Define the way that ensemble must compute the predictions. The default valid options are:
  - 'avg' Compute the mean of probabilities and the bipartitions
  - 'maj' Compute the majority of votes
  - 'max' Compute the higher probability for each instance/label
  - 'min' Compute the lower probability for each instance/label
  (Default: 'maj')
- **probability** A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1.
create_holdout_partition

**Description**

This method creates multi-label dataset for train, test, validation or other proposes the partition method defined in `method`. The number of partitions is defined in `partitions` parameter. Each instance is used in only one partition of division.

---

**Value**

A `mlresult` with computed predictions.

**Functions**

- `utiml_predict_ensemble`: Internal version

**Note**

You can create your own vote schema, just create a method that receive two matrix (bipartitions and probabilities) and return a list with the final bipartitions and probabilities.

Remember that this method will compute the ensemble votes for each label. Thus the bipartition and probability matrix passed as argument for this method is related with the bipartitions and probabilities for a single label.

**Examples**

```r
## Not run:
model <- br(toyml, "KNN")
predictions <- list(
  predict(model, toyml[1:10], k=1),
  predict(model, toyml[1:10], k=3),
  predict(model, toyml[1:10], k=5)
)
result <- compute_multilabel_predictions(predictions, "maj")

## Random choice
random_choice <- function(bipartition, probability) {
  cols <- sample(seq(ncol(bipartition)), nrow(bipartition), replace = TRUE)
  list(
    bipartition = bipartition[, cbind(seq(nrow(bipartition)), cols)],
    probability = probability[, cbind(seq(nrow(probability)), cols)]
  )
}
result <- compute_multilabel_predictions(predictions, "random_choice")

## End(Not run)
```
create_holdout_partition

Usage

create_holdout_partition(mdata, partitions = c(train = 0.7, test = 0.3), method = c("random", "iterative", "stratified"))

Arguments

mdata
A mlr dataset.

partitions
A list of percentages or a single value. The sum of all values does not be greater than 1. If a single value is informed then the complement of them is applied to generated the second partition. If two or more values are informed and the sum of them is lower than 1 the partitions will be generated with the informed proportion. If partitions have names, they are used to name the return. (Default: c(train=0.7, test=0.3)).

method
The method to split the data. The default methods are:

- random Split randomly the folds.
- iterative Split the folds considering the labels proportions individually. Some specific label can not occurs in all folds.
- stratified Split the folds considering the labelset proportions.

You can also create your own partition method. See the note and example sections to more details. (Default: "random")

Value

A list with at least two datasets sampled as specified in partitions parameter.

Note

To create your own split method, you need to build a function that receive a mlr object and a list with the proportions of examples in each fold and return an other list with the index of the elements for each fold.

References


See Also

Other sampling: create_kfold_partition, create_random_subset, create_subset

Examples

dataset <- create_holdout_partition(toyml)
names(dataset)
# [1] "train" "test"
#dataset$train
#dataset$test
dataset <- create_holdout_partition(toyml, c(a=0.1, b=0.2, c=0.3, d=0.4))
# names(dataset)
# [[1] "a" "b" "c" "d"

sequential_split <- function(mdata, r) {
  S <- list()

  amount <- trunc(r * mdata$measures$numinstances)
  indexes <- c(0, cumsum(amount))
  indexes[length(r)+1] <- mdata$measures$numinstances

  S <- lapply(seq(length(r)), function(i) {
    seq(indexes[i]+1, indexes[i+1])
  })

  S
}
dataset <- create_holdout_partition(toyml, method="sequential_split")

---

create_kfold_partition

Create the k-folds partition based on the specified algorithm

---

Description

This method create the kFoldPartition object, from it is possible create the dataset partitions to train, test and optionally to validation.

Usage

create_kfold_partition(mdata, k = 10, method = c("random", "iterative", "stratified"))

Arguments

- mdata: A mldr dataset.
- k: The number of desirable folds. (Default: 10)
- method: The method to split the data. The default methods are:
  - **random**: Split randomly the folds.
  - **iterative**: Split the folds considering the labels proportions individually. Some specific label can not occurs in all folds.
  - **stratified**: Split the folds considering the labelset proportions.

You can also create your own partition method. See the note and example sections to more details. (Default: "random")
create_random_subset

Value

An object of type kFoldPartition.

Note

To create your own split method, you need to build a function that receive a mldr object and a list with the proportions of examples in each fold and return an other list with the index of the elements for each fold.

References


See Also

How to create the datasets from folds

Other sampling: create_holdout_partition, create_random_subset, create_subset

Examples

```r
k10 <- create_kfold_partition(toyml, 10)
k5 <- create_kfold_partition(toyml, 5, "stratified")

sequential_split <- function (mdata, r) {
  S <- list()

  amount <- trunc(r * mdata$measures$num.instances)
  indexes <- c(0, cumsum(amount))
  indexes[length(r)+1] <- mdata$measures$num.instances

  S <- lapply(seq(length(r)), function (i) {
    seq(indexes[i]+1, indexes[i+1])
  })

  S
}

k3 <- create_kfold_partition(toyml, 3, "sequential_split")
```

create_random_subset

Create a random subset of a dataset

Description

Create a random subset of a dataset
create_subset

Usage

create_random_subset(mdata, instances, attributes = mdata$measures$num.inputs, replacement = FALSE)

Arguments

mdata A mldr dataset
instances The number of expected instances
attributes The number of expected attributes. (Default: all attributes)
replacement A boolean value to define sample with replacement or not. (Default: FALSE)

Value

A new mldr subset

See Also

Other sampling: create_holdout_partition, create_kfold_partition, create_subset

Examples

small.toy <- create_random_subset(toyml, 10, 3)
medium.toy <- create_random_subset(toyml, 50, 5)

create_subset

Create a subset of a dataset

Description

Create a subset of a dataset

Usage

create_subset(mdata, rows, cols = NULL)

Arguments

mdata A mldr dataset
rows A vector with the instances indexes (names or indexes).
cols A vector with the attributes indexes (names or indexes).

Value

A new mldr subset
Note
It is not necessary specify the labels attributes because they are included by default.

See Also
Other sampling: create_holdout_partition, create_kfold_partition, create_random_subset

Examples

```r
## Create a dataset with the 20 first examples and the 7 first attributes
small.toy <- create_subset(toyml, seq(20), seq(7))

## Create a random dataset with 50 examples and 5 attributes
random.toy <- create_subset(toyml, sample(100, 50), sample(10, 5))
```

Description

Create a binary relevance with ConTRolled Label correlation exploitation (CTRL) model for mul-
tilabel classification.

Usage

```r
ctrl(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
    m = 5, validation.size = 0.3, validation.threshold = 0.3, ..., 
    predict.params = list(), cores = getOption("utiml.cores", 1),
    seed = getOption("utiml.seed", NA))
```

Arguments

- `mdata`: A mldr dataset used to train the binary models.
- `base.algorithm`: A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM")
- `m`: The max number of Binary Relevance models used in the binary ensemble. (De-
default: 5)
- `validation.size`: The size of validation set, used internally to prunes error-prone class labels. The
  value must be between 0.1 and 0.5. (Default: 0.3)
- `validation.threshold`: Thresholding parameter determining whether any class label in Y is regarded as
  error-prone or not. (Default: 0.3)
- `...`: Others arguments passed to the base algorithm for all subproblems
- `predict.params`: A list of default arguments passed to the predictor algorithm. (default: list())
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the
  parallel package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run
  in parallel. (Default: options("utiml.seed", NA))
Details
CTRL employs a two-stage filtering procedure to exploit label correlations in a controlled manner. In the first stage, error-prone class labels are pruned from Y to generate the candidate label set for correlation exploitation. In the second stage, classification models are built for each class label by exploiting its closely-related labels in the candidate label set.

Dependencies: The degree of label correlations are estimated via supervised feature selection techniques. Thus, this implementation use the relief method available in FSelector package.

Value
An object of class ctrlmodel containing the set of fitted models, including:

- **rounds**: The value passed in the m parameter
- **validation.size**: The value passed in the validation.size parameter
- **validation.threshold**: The value passed in the validation.threshold parameter
- **Y**: Name of labels less susceptible to error, according to the validation process
- **R**: List of close-related labels related with Y obtained by using feature selection technique
- **models**: A list of the generated models, for each label a list of models was built based on close-related labels.

References
Li, Y., & Zhang, M. (2014). Enhancing Binary Relevance for Multi-label Learning with Controlled Label Correlations Exploitation. In 13th Pacific Rim International Conference on Artificial Intelligence (pp. 91-103). Gold Coast, Australia.

See Also
Other Transformation methods: brplus, br, cc, clr, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc

Examples
```r
model <- ctrl(toyml, "RANDOM")
pred <- predict(model, toyml)

# Not run:
# Change default values and use 4 CORES
model <- ctrl(toyml, 'C5.0', m = 10, validation.size = 0.4,
validation.threshold = 0.5, cores = 4)

# Use seed
model <- ctrl(toyml, 'RF', cores = 4, seed = 123)

# Set a parameters for all subproblems
model <- ctrl(dataset$train, 'KNN', k=5)

# End(Not run)
```
**Description**

Create a DBR classifier to predict multi-label data. This is a simple approach that enables the binary classifiers to discover existing label dependency by themselves. The idea of DBR is exactly the same used in BR+ (the training method is the same, excepted by the argument `estimate.models` that indicate if the estimated models must be created).

**Usage**

```r
dbr(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
    estimate.models = TRUE,..., cores = getOption("utiml.cores", 1),
    seed = getOption("utiml.seed", NA))
```

**Arguments**

- `mdata` A mlr dataset used to train the binary models.
- `base.algorithm` A string with the name of the base algorithm. (Default: `options("utiml.base.algorithm", "SVM")`)
- `estimate.models` Logical value indicating whether is necessary build Binary Relevance classifier for estimate process. The default implementation use BR as estimators, however when other classifier is desirable then use the value `FALSE` to skip this process. (Default: `TRUE`).
- `...` Others arguments passed to the base algorithm for all subproblems.
- `cores` The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `options("utiml.cores", 1)`)  
- `seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `options("utiml.seed", NA)`)  

**Value**

An object of class `DBRmodel` containing the set of fitted models, including:

- `labels` A vector with the label names.
- `estimation` The BR model to estimate the values for the labels. Only when the `estimate.models = TRUE`.
- `models` A list of final models named by the label names.

**References**

See Also

Recursive Dependent Binary Relevance

Other Transformation methods: brplus, br, cc, clr, ctrl, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc

Examples

```r
model <- dbr(toyml, "RANDOM")
pred <- predict(model, toyml)

# Not run:
# Use Random Forest as base algorithm and 4 cores
model <- dbr(toyml, 'RF', cores = 4)

# End(Not run)
```

ebr  

**Ensemble of Binary Relevance for multi-label Classification**

Description

Create an Ensemble of Binary Relevance model for multilabel classification.

Usage

```r
ebr(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
   m = 10, subsample = 0.75, attr.space = 0.5, replacement = TRUE, ..., 
   cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

Arguments

- `mdata` A mlr dataset used to train the binary models.
- `base.algorithm` A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
- `m` The number of Binary Relevance models used in the ensemble. (Default: 10)
- `subsample` A value between 0.1 and 1 to determine the percentage of training instances that must be used for each classifier. (Default: 0.75)
- `attr.space` A value between 0.1 and 1 to determine the percentage of attributes that must be used for each classifier. (Default: 0.50)
- `replacement` Boolean value to define if use sampling with replacement to create the data of the models of the ensemble. (Default: TRUE)
- `cores` The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: options("utiml.cores", 1))
- `seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))
Details

This model is composed by a set of Binary Relevance models. Binary Relevance is a simple and effective transformation method to predict multi-label data.

Value

An object of class `ebrmodel` containing the set of fitted BR models, including:

- `models` A list of BR models.
- `nrow` The number of instances used in each training dataset.
- `ncol` The number of attributes used in each training dataset.
- `rounds` The number of interactions.

Note

If you want to reproduce the same classification and obtain the same result will be necessary set a flag `utiml.mc.set.seed` to FALSE.

References


See Also

Other Transformation methods: `brplus`, `br`, `cc`, `clr`, `ctrl`, `dbr`, `ecc`, `eps`, `homer`, `lift`, `lp`, `mbr`, `ns`, `ppt`, `prudent`, `ps`, `rakel`, `rdbr`, `rpc`

Other Ensemble methods: `ecc`, `eps`

Examples

```r
model <- ebr(toyml, "RANDOM")
pred <- predict(model, toyml)
```

```r
# Not run:
# Use J48 with 90% of instances and only 5 rounds
model <- ebr(toyml, 'J48', m = 5, subsample = 0.9)

# Use 75% of attributes
model <- ebr(dataset$train, attr.space = 0.75)

# Running in 4 cores and define a specific seed
model1 <- ebr(toyml, cores=4, seed = 312)
```

```r
## End(Not run)
```
Ensemble of Classifier Chains for multi-label Classification

Description

Create an Ensemble of Classifier Chains model for multilabel classification.

Usage

ecc(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
   m = 10, subsample = 0.75, attr.space = 0.5, replacement = TRUE, ..., 
   cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))

Arguments

mdata A mlr dataset used to train the binary models.
base.algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
m The number of Classifier Chains models used in the ensemble. (Default: 10)
subsample A value between 0.1 and 1 to determine the percentage of training instances that 
must be used for each classifier. (Default: 0.75)
attr.space A value between 0.1 and 1 to determine the percentage of attributes that must be 
used for each classifier. (Default: 0.50)
replacement Boolean value to define if use sampling with replacement to create the data of 
the models of the ensemble. (Default: TRUE)
... Others arguments passed to the base algorithm for all subproblems.
cores The number of cores to parallelize the training. Values higher than 1 require the 
parallel package. (Default: options("utiml.cores", 1))
seed An optional integer used to set the seed. This is useful when the method is run 
in parallel. (Default: options("utiml.seed", NA))

Details

This model is composed by a set of Classifier Chains models. Classifier Chains is a Binary Relevance transformation method based to predict multi-label data. It is different from BR method due the strategy of extended the attribute space with the 0/1 label relevances of all previous classifiers, forming a classifier chain.

Value

An object of class ECCmodel containing the set of fitted CC models, including:

rounds The number of interactions
models A list of BR models.
nrow The number of instances used in each training dataset
ncol The number of attributes used in each training dataset
Note
If you want to reproduce the same classification and obtain the same result will be necessary set a flag utiml.mc.set.seed to FALSE.

References

See Also
Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, raket, rdbr, rpc
Other Ensemble methods: ebr, eps

Examples

```r
# Use all default values
model <- ecc(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use J48 with 100% of instances and only 5 rounds
model <- ecc(toyml, 'J48', m = 5, subsample = 1)

# Use 75% of attributes
model <- ecc(toyml, attr.space = 0.75)

# Running in 4 cores and define a specific seed
modell <- ecc(toyml, cores=4, seed=123)

## End(Not run)
```

Description
Create an Ensemble of Pruned Set model for multilabel classification.

Usage
```r
eps(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
m = 10, subsample = 0.75, p = 3, strategy = c("A", "B"), b = 2, ...
cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```
Arguments

- mdata: A mlr dataset used to train the binary models.
- base.algorithm: A string with the name of the base algorithm. (Default: options("utiml.base.algorith"m", "SVM"))
- m: The number of Pruned Set models used in the ensemble.
- subsample: A value between 0.1 and 1 to determine the percentage of training instances that must be used for each classifier. (Default: 0.63)
- p: Number of instances to prune. All labelsets that occurs p times or less in the training data is removed. (Default: 3)
- strategy: The strategy (A or B) for processing infrequent labelsets. (Default: A).
- b: The number used by the strategy for processing infrequent labelsets.
- ...: Others arguments passed to the base algorithm for all subproblems.
- cores: The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores"", 1))
- seed: An optional integer used to set the seed. (Default: options("utiml.seed"", NA))

Details

Pruned Set (PS) is a multi-class transformation that remove the less common classes to predict multi-label data. The ensemble is created with different subsets of the original multi-label data.

Value

An object of class EPSmodel containing the set of fitted models, including:

- rounds: The number of interactions
- models: A list of PS models.

References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc
Other Powerset: lp, ppt, ps, rakel
Other Ensemble methods: ebr, ecc

Examples

```r
model <- eps(toyml, "RANDOM")
pred <- predict(model, toyml)
```

## Not run:
## Change default configurations
fill_sparse_mldata

```r
model <- eps(toyml, "RF", m=15, subsample=0.4, p=4, strategy="B", b=4)
## End(Not run)
```

fill_sparse_mldata Fill sparse dataset with 0 or " values

Description
Transform a sparse dataset filling NA values to 0 or " based on the column type. Text columns with numeric values will be modified to numerical.

Usage
```
fill_sparse_mldata(mdata)
```

Arguments
```
mdata The mldr dataset to be filled.
```

Value
a new mldr object.

See Also
Other pre process: normalize_mldata, remove_attributes, remove_labels, remove skewness_labels, remove_unique_attributes, remove_unlabeled_instances, replace_nominal_attributes

Examples
```
sparse.toy <- toyml
sparse.toy$dataset$ratt10[sample(100, 30)] <- NA
complete.toy <- fill_sparse_mldata(sparse.toy)
```

fixed_threshold Apply a fixed threshold in the results

Description
Transform a prediction matrix with scores/probabilities in a mlresult applying a fixed threshold. A global fixed threshold can be used of all labels or different fixed thresholds, one for each label.
Usage

```r
fixed_threshold(prediction, threshold = 0.5, probability = FALSE)
```

## Default S3 method:
```r
fixed_threshold(prediction, threshold = 0.5,
                 probability = FALSE)
```

## S3 method for class 'mlresult'
```r
fixed_threshold(prediction, threshold = 0.5,
                 probability = FALSE)
```

Arguments

- **prediction**: A matrix with scores/probabilities where the columns are the labels and the rows are the instances.
- **threshold**: A single value between 0 and 1 or a list with threshold values contained one value per label.
- **probability**: A logical value. If `TRUE` the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: `FALSE`)

Value

A `mlresult` object.

Methods (by class)

- **default**: Fixed Threshold for matrix or data.frame
- **mlresult**: Fixed Threshold for `mlresult`

References


See Also

Other threshold: `lcard_threshold, mcut_threshold, pcut_threshold, rcut_threshold, scut_threshold, subset_correction`

Examples

```r
# Create a prediction matrix with scores
result <- matrix(
  data = rnorm(9, 0.5, 0.2),
  ncol = 3,
  dimnames = list(NULL, c('lb1', 'lb2', 'lb3'))
)
```
# Use 0.5 as threshold
fixed_threshold(result)

# Use an threshold for each label
fixed_threshold(result, c(0.4, 0.6, 0.7))

homer

Hierarchy Of Multilabel classifier (HOMER)

Description

Create a Hierarchy Of Multilabel classifier (HOMER).

Usage

```r
homer(mdata, base_algorithm = getOption("utiml.base.algorithm", "SVM"),
clusters = 3, method = c("balanced", "clustering", "random"),
iteration = 100L, ..., cores = getOption("utiml.cores", 1),
seed = getOption("utiml.seed", NA))
```

Arguments

- `mdata`: A mlr dataset used to train the binary models.
- `base_algorithm`: A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
- `clusters`: Number maximum of nodes in each level. (Default: 3)
- `method`: The strategy used to organize the labels (create the meta-labels). The options are: "balanced", "clustering" and "random". (Default: "balanced").
- `iteration`: The number max of iterations, used by balanced or clustering methods.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Details

HOMER is an algorithm for effective and computationally efficient multilabel classification in domains with many labels. It constructs a hierarchy of multilabel classifiers, each one dealing with a much smaller set of labels.

Value

An object of class `homermodel` containing the set of fitted models, including:

- `labels`: A vector with the label names.
- `clusters`: The number of nodes in each level
- `models`: The Hierarchy of BR models.
is.bipartition

Test if a mlresult contains crisp values as default

Description

Test if a mlresult contains crisp values as default

Usage

is.bipartition(mlresult)

Arguments

mlresult The mlresult object

Value

logical value
is.probability  

Test if a mlresult contains score values as default

Description
Test if a mlresult contains score values as default

Usage
is.probability(mlresult)

Arguments
mlresult The mlresult object

Value
logical value

lcard_threshold

Threshold based on cardinality

Description
Find and apply the best threshold based on cardinality of training set. The threshold is choice based on how much the average observed label cardinality is close to the average predicted label cardinality.

Usage
lcard_threshold(prediction, cardinality, probability = FALSE)

## Default S3 method:
lcard_threshold(prediction, cardinality,
 probability = FALSE)

## S3 method for class 'mlresult'
lcard_threshold(prediction, cardinality,
 probability = FALSE)

Arguments
prediction A matrix or mlresult.
cardinality A real value of training dataset label cardinality, used to define the threshold value.
probability A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: FALSE)
Value

A mlresult object.

Methods (by class)

- default: Cardinality Threshold for matrix or data.frame
- mlresult: Cardinality Threshold for mlresult

References


See Also

Other threshold: fixed_threshold, mcut_threshold, pcut_threshold, rcut_threshold, scut_threshold, subset_correction

Examples

```r
prediction <- matrix(runif(16), ncol = 4)
lcard_threshold(prediction, 2.1)
```

--

lift  

LIFT for multi-label Classification

Description

Create a multi-label learning with Label specific Features (LIFT) model.

Usage

```r
lift(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
    ratio = 0.1, ..., cores = getOption("utiml.cores", 1),
    seed = getOption("utiml.seed", NA))
```

Arguments

- `mdata`: A mlr dataset used to train the binary models.
- `base.algorithm`: A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
- `ratio`: Control the number of clusters being retained. Must be between 0 and 1. (Default: 0.1)
- `...`: Others arguments passed to the base algorithm for all subproblems.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))
Details

LIFT firstly constructs features specific to each label by conducting clustering analysis on its positive and negative instances, and then performs training and testing by querying the clustering results.

Value

An object of class LIFTmodel containing the set of fitted models, including:

- **labels**: A vector with the label names.
- **models**: A list of the generated models, named by the label names.

References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lp, mbr, ns, ppt, prudent, ps, rakel, rdb, rpc

Examples

```r
model <- lift(toyml, "RANDOM")
pred <- predict(model, toyml)

# Not run:
# Running lift with a specific ratio
model <- lift(toyml, "RF", 0.15)

# End(Not run)
```

---

**lp**

*Label Powerset for multi-label Classification*

Description

Create a Label Powerset model for multilabel classification.

Usage

```r
lp(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```
Arguments

- mdata: A mldr dataset used to train the binary models.
- base.algorithm: A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
- ...: Others arguments passed to the base algorithm for all subproblems
- cores: Not used
- seed: An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Details

Label Powerset is a simple transformation method to predict multi-label data. This is based on the multi-class approach to build a model where the classes are each labelset.

Value

An object of class LPmodel containing the set of fitted models, including:

- labels: A vector with the label names.
- model: A multi-class model.

References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, mbr, ns, ppt, prudent, ps, rakel, rdbr, rpc

Other Powerset: eps, ppt, ps, rakel

Examples

```r
model <- lp(toyml, "RANDOM")
pred <- predict(model, toyml)
```

mbr

Meta-BR or 2BR for multi-label Classification

Description

Create a Meta-BR (MBR) classifier to predict multi-label data. To this, two round of Binary Relevance is executed, such that, the first step generates new attributes to enrich the second prediction.
Usage

```r
mbr(mdata, base.algorithm = getOption("umlr.base.algorithm", "SVM"),
    folds = 1, phi = 0, ..., predict.params = list(),
    cores = getOption("umlr.cores", 1), seed = getOption("umlr.seed", NA))
```

Arguments

- **mdata**: A mldr dataset used to train the binary models.
- **base.algorithm**: A string with the name of the base algorithm. (Default: options("umlr.base.algorithm", "SVM"))
- **folds**: The number of folds used in internal prediction. If this value is 1 all dataset will
  be used in the first prediction. (Default: 1)
- **phi**: A value between 0 and 1 to determine the correlation coefficient. The value 0
  include all labels in the second phase and the 1 only the predicted label. (Default: 0)
- **...**: Others arguments passed to the base algorithm for all subproblems.
- **predict.params**: A list of default arguments passed to the predictor algorithm. (Default: list())
- **cores**: The number of cores to parallelize the training. Values higher than 1 require the
  parallel package. (Default: options("umlr.cores", 1))
- **seed**: An optional integer used to set the seed. This is useful when the method is run
  in parallel. (Default: options("umlr.seed", NA))

Details

This implementation use complete training set for both training and prediction steps of 2BR. However, the phi parameter may be used to remove labels with low correlations on the second step.

Value

An object of class `MBrmodel` containing the set of fitted models, including:

- **labels**: A vector with the label names.
- **phi**: The value of phi parameter.
- **correlation**: The matrix of label correlations used in combination with phi parameter to define the labels used in the second step.
- **basemodel**: The BRModel used in the first iteration.
- **models**: A list of models named by the label names used in the second iteration.

References

See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, ns, ppt, prudent, ps, rakel, rdbr, rpc

Other Stacking methods: brplus

Examples

```r
model <- mbr(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use 10 folds and different phi correlation with J48 classifier
model <- mbr(toyml, 'J48', 10, 0.2)

# Run with 4 cores
model <- mbr(toyml, "SVM", cores = 4, seed = 123)

# Set a specific parameter
model <- mbr(toyml, 'KNN', k=5)

## End(Not run)
```

---

**mcut_threshold**

**Maximum Cut Thresholding (MCut)**

**Description**

The Maximum Cut (MCut) automatically determines a threshold for each instance that selects a subset of labels with higher scores than others. This leads to the selection of the middle of the interval defined by these two scores as the threshold.

**Usage**

```r
mcut_threshold(prediction, probability = FALSE)
```

## Default S3 method:

```r
mcut_threshold(prediction, probability = FALSE)
```

## S3 method for class 'mlresult'

```r
mcut_threshold(prediction, probability = FALSE)
```

**Arguments**

- `prediction` A matrix or mlresult.
- `probability` A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: FALSE)
*merge_mlconfmat*

**Value**

A mlresult object.

**Methods (by class)**

- **default**: Maximum Cut Thresholding (MCut) method for matrix
- **mlresult**: Maximum Cut Thresholding (MCut) for mlresult

**References**


**See Also**

Other threshold: `fixed_threshold`, `lcard_threshold`, `pcut_threshold`, `rcut_threshold`, `scut_threshold`, `subset_correction`

**Examples**

```r
prediction <- matrix(runif(16), ncol = 4)
mcut_threshold(prediction)
```

---

**merge_mlconfmat**  
*Join a list of multi-label confusion matrix*

**Description**

Join a list of multi-label confusion matrix

**Usage**

```r
merge_mlconfmat(object, ...)
```

**Arguments**

- `object`  
  A mlconfmat object or a list of mlconfmat objects
- `...`  
  mlconfmat objects

**Value**

mlconfmat
mldata

*Fix the mldr dataset to use factors*

**Description**

Fix the mldr dataset to use factors

**Usage**

```r
mldata(mdata)
```

**Arguments**

- `mdata` A mldr dataset.

**Value**

A mldr object

**Examples**

```r
toym1 <- mldata(toym1)
```

mlpredict

*Prediction transformation problems*

**Description**

Base classifiers are used to build models to solve the transformation problems. To create a new base classifier, two steps are necessary:

1. Create a train method
2. Create a prediction method

This section is about how to create the second step: a prediction method. To create a new train method see `mltrain` documentation.

**Usage**

```r
mlpredict(model, newdata, ...)
```

## Default S3 method:
```r
mlpredict(model, newdata, ...)
```

## S3 method for class 'svm'
```r
mlpredict(model, newdata, ...)
```
## Arguments

- **model**: An object model returned by some mltrain method, its class determine the name of this method.
- **newdata**: A data.frame with the new data to be predicted.
- **...**: Others arguments passed to the predict method.

## Value

A matrix with the probabilities of each class value/example, where the rows are the examples and the columns the class values.

## Methods (by class)

- **default**: Default S3 method

```r
## S3 method for class 'SMO'
mlpredict(model, newdata, ...)

## S3 method for class 'J48'
mlpredict(model, newdata, ...)

## S3 method for class 'C5.0'
mlpredict(model, newdata, ...)

## S3 method for class 'rpart'
mlpredict(model, newdata, ...)

## S3 method for class 'randomForest'
mlpredict(model, newdata, ...)

## S3 method for class 'naiveBayes'
mlpredict(model, newdata, ...)

## S3 method for class 'baseKNN'
mlpredict(model, newdata, ...)

## S3 method for class 'xgb.Booster'
mlpredict(model, newdata, ...)

## S3 method for class 'majorityModel'
mlpredict(model, newdata, ...)

## S3 method for class 'randomModel'
mlpredict(model, newdata, ...)

## S3 method for class 'emptyModel'
mlpredict(model, newdata, ...)
```
• svm: SVM implementation (require e1071 package to use)
• smo: SMO implementation (require RWeka package to use)
• J48: J48 implementation (require RWeka package to use)
• C5.0: C5.0 implementation (require C50 package to use)
• rpart: CART implementation (require rpart package)
• randomForest: Random Forest (RF) implementation (require randomForest package to use)
• naiveBayes: Naive Bayes (NB) implementation (require e1071 package to use)
• baseKNN: kNN implementation (require kknn package to use)
• xgb.Booster: XGBoost implementation (require xgboost package)
• majorityModel: Majority prediction
• randomModel: Majority prediction
• emptyModel: Empty model to fix the cases with few train examples

How to create a new prediction base method

First is necessary to know the class of model generate by the respective train method, because this name determines the method name. It must start with 'mlpredict.', followed by the model class name, e.g. a model with class 'fooModel' must be called as mlpredict.fooModel.

After defined the name, you need to implement your prediction base method. The model built on mltrain is available on model parameter and the newdata is the data to be predict.

The return of this method must be a data.frame with two columns called "prediction" and "probability". The first column contains the predicted class and the second the probability/score/confidence of this prediction. The rows represents the examples.

Examples

# Create a method that predict always the first class
# The model must be of the class 'fooModel'
mlpredict.fooModel <- function (model, newdata, ...) {
  # Predict the first class with a random confidence
  data.frame(
    prediction = rep(model$classes[1], nrow(newdata)),
    probability = sapply(runif(nrow(newdata)), function (score) {
      max(score, 1 - score)
    } ),
    row.names = rownames(newdata)
  )
}

# Not run:
# Create a SVM predict method using the e1071 package (the class of SVM model
# from e1071 package is 'svm')
library(e1071)
mlpredict.svm <- function (dataset, newdata, ...) {
  result <- predict(model, newdata, probability = TRUE, ...)
  attr(result, 'probabilities')
}
mltrain  Build transformation models

Description

Base classifiers are used to build models to solve the transformation problems. To create a new base classifier, two steps are necessary:

1. Create a train method
2. Create a prediction method

This section is about how to create the first step: a train method. To create a new predict model see mlpredict documentation.

Usage

mltrain(object, ...)

## Default S3 method:
mltrain(object, ...)

## S3 method for class 'baseSVM'
mltrain(object, ...)

## S3 method for class 'baseSMO'
mltrain(object, ...)

## S3 method for class 'baseJ48'
mltrain(object, ...)

## S3 method for class 'baseC5.0'
mltrain(object, ...)

## S3 method for class 'baseCART'
mltrain(object, ...)

## S3 method for class 'baseRF'
mltrain(object, ...)

## S3 method for class 'baseNB'
mltrain(object, ...)

## S3 method for class 'baseKNN'
mltrain(object, ...)

## S3 method for class 'baseXGB'
mltrain(object, ...)

## S3 method for class 'baseMAJORITY'
mltrain(object, ...)

## S3 method for class 'baseRANDOM'
mltrain(object, ...)

**Arguments**

- **object** A mltransformation object. This is used as a list and contains at least five values:
  - **object$data** A data.frame with the train data, where the columns are the attributes and the rows are the examples.
  - **object$labelname** The name of the class column.
  - **object$labelindex** The column index of the class.
  - **object$mldataset** The name of multi-label dataset.
  - **object$mlmethod** The name of the multi-label method.
  - Others values may be specified by the multi-label method.

- **...** Others arguments passed to the base method.

**Value**

A model object. The class of this model can be of any type, however, this object will be passed to the respective mlpredict method.

**Methods (by class)**

- **default**: Default S3 method
- **baseSVM**: SVM implementation (require e1071 package to use)
- **baseSMO**: SMO implementation (require RWeka package to use)
- **baseJ48**: J48 implementation (require RWeka package to use)
- **baseC5.0**: C5.0 implementation (require C50 package to use)
- **baseCART**: CART implementation (require rpart package to use)
- **baseRF**: Random Forest (RF) implementation (require randomForest package to use)
- **baseNB**: Naive Bayes (NB) implementation (require e1071 package to use)
- **baseKNN**: kNN implementation (require kknn package to use)
- **baseXGB**: XGBoost implementation (require xgboost package)
- **baseMAJORITY**: Majority model
- **baseRANDOM**: Random model
How to create a new train base method

First, it is necessary to define a name of your classifier, because this name determines the method name. The base method name must start with mltrain.base followed by the designed name, e.g. a 'FOO' classify must be defined as mltrain.baseFOO (we suggest always use upper case names).

Next, your method must receive at least two parameters (object, ...). Use object$data[, object$labelindex] or object$data[, object$labelname] to access the labels values and use object$data[, ~object$labelindex] to access the predictive attributes. If you need to know which are the multi-label dataset and method, use object$mldataset and object$mlmethod, respectively.

Finally, your method should return a model that will be used by the mlpredict method. Remember, that your method may be used to build binary and multi-class models.

Examples

# Create a empty model of type FOO
mltrain.baseFOO <- function (object, ...) {
  mymodel <- list(
    classes = as.character(unique(object$data[, object$labelindex]))
  )
  class(mymodel) <- 'FooModel'
  mymodel
}

# Using this base method with Binary Relevance
brmodel <- br(toyml, 'FOO')

## Not run:

# Create a SVM method using the e1071 package
library(e1071)
mltrain.baseSVM <- function (object, ...) {
  traindata <- object$data[, ~object$labelindex]
  labeldata <- object$data[, object$labelindex]
  model <- svm(traindata, labeldata, probability = TRUE, ...)
  model
}

## End(Not run)

---

multilabel_confusion_matrix

Compute the confusion matrix for a multi-label prediction

Description

The multi-label confusion matrix is an object that contains the prediction, the expected values and also a lot of pre-processed information related with these data.
Usage

`multilabel_confusion_matrix(mdata, mlresult)`

Arguments

- `mdata` A mldr dataset
- `mlresult` A mlresult prediction

Value

A mlconfmat object that contains:

- `Z` The bipartition matrix prediction.
- `Fx` The score/probability matrix prediction.
- `R` The ranking matrix prediction.
- `Y` The expected matrix bipartition.
- `TP` The True Positive matrix values.
- `FP` The False Positive matrix values.
- `TN` The True Negative matrix values.
- `FN` The False Negative matrix values.
- `Zi` The total of positive predictions for each instance.
- `Yi` The total of positive expected for each instance.
- `TPi` The total of True Positive predictions for each instance.
- `FPi` The total of False Positive predictions for each instance.
- `TNi` The total of True Negative predictions for each instance.
- `FNi` The total False Negative predictions for each instance.
- `Zl` The total of positive predictions for each label.
- `Yl` The total of positive expected for each label.
- `TPl` The total of True Positive predictions for each label.
- `FPl` The total of False Positive predictions for each label.
- `TNl` The total of True Negative predictions for each label.
- `FNl` The total False Negative predictions for each label.

Examples

```r
## Not run:
prediction <- predict(br(toyml), toyml)
mlconfmat <- multilabel_confusion_matrix(toyml, prediction)

# Label with the most number of True Positive values
which.max(mlconfmat$TPl)
```
multilabel_evaluate

Evaluate multi-label predictions

Description

This method is used to evaluate multi-label predictions. You can create a confusion matrix object or use directly the test dataset and the predictions. You can also specify which measures do you desire use.

Usage

```
multilabel_evaluate(object, ...)  
```

```
# S3 method for class 'mldr'
multilabel_evaluate(object, mlresult, measures = c("all"),
                     labels = FALSE, ...)
```

```
# S3 method for class 'mlconfmat'
multilabel_evaluate(object, measures = c("all"),
                     labels = FALSE, ...)
```

Arguments

- **object**: A mldr dataset or a mlconfmat confusion matrix
- **...**: Extra parameters to specific measures.
- **mlresult**: The prediction result (Optional, required only when the mldr is used).
- **measures**: The measures names to be computed. Call `multilabel_measures()` to see the expected measures. You also can use "bipartition", "ranking", "label-based", "example-based", "macro-based" and "micro-based" to include a set of measures. (Default: "all").
- **labels**: Logical value defining if the label results should be also returned. (Default: FALSE)
multilabel_measures

Value

If labels is FALSE return a vector with the expected multi-label measures, otherwise, a list contained the multi-label and label measures.

Methods (by class)

- `mldr`: Default S3 method
- `mlconfmat`: Default S3 method

References


Examples

```r
## Not run:
prediction <- predict(benchmark(toyml), toyml)

# Compute all measures
multilabel_evaluate(toyml, prediction)

# Compute bipartition measures
multilabel_evaluate(toyml, prediction, "bipartition")

# Compute multilabel measures
multilabel_evaluate(toyml, prediction, c("accuracy", "F1", "macro-based"))

# Compute the confusion matrix before the measures
cm <- multilabel_confusion_matrix(toyml, prediction)
multilabel_evaluate(cm, "example-based")
multilabel_evaluate(cm, c("hamming-loss", "subset-accuracy", "F1"))

## End(Not run)
```

---

multilabel_measures  
Return the name of all measures

Description

Return the name of all measures
**Usage**

```
multilabel_measures()
```

**Value**

array of character contained the measures names.

**Examples**

```
multilabel_measures()
```

---

**multilabel_prediction**  
*Create a mlresult object*

**Description**

Create a mlresult object

**Usage**

```
multilabel_prediction(bipartitions, probabilities,
    probability = getOption("utiml.use.probs", TRUE),
    empty.prediction = getOption("utiml.empty.prediction", FALSE))
```

**Arguments**

- `bipartitions`: The matrix of predictions (bipartition values), only 0 and 1
- `probabilities`: The matrix of probability/confidence of a prediction, between 0..1
- `probability`: A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: `getOption("utiml.use.probs", TRUE)`)
- `empty.prediction`: A logical value. If TRUE the predicted values may contains empty values, otherwise at least one label will be positive for each instance.

**Value**

An object of type mlresult

**Examples**

```
probs <- matrix(
    runif(90), ncol=3, dimnames = list(1:30, c("y1", "y2", "y3"))
)
preds <- matrix(
    as.numeric(probs > 0.5), ncol=3, dimnames = list(1:30, c("y1", "y2", "y3"))
)
multilabel_prediction(probs, preds)
```
normalize_mldata Normalize numerical attributes

Description

Normalize all numerical attributes to values between 0 and 1. The highest value is changed to 1 and the lowest value to 0.

Usage

normalize_mldata(mdata)

Arguments

mdata The mldr dataset to be normalized.

Value

a new mldr object.

See Also

Other pre process: fill_sparse_mldata, remove_attributes, remove_labels, remove_skewness_labels, remove_unique_attributes, remove_unlabeled_instances, replace_nominal_attributes

Examples

norm.toy <- normalize_mldata(toyml)

ns Nested Stacking for multi-label Classification

Description

Create a Nested Stacking model for multilabel classification.

Usage

ns(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"), chain = NA, ..., predict.params = list(), cores = NULL, seed = getOption("utiml.seed", NA))
Arguments

mdata A mlr dataset used to train the binary models.

base_algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))

chain A vector with the label names to define the chain order. If empty the chain is the default label sequence of the dataset. (Default: NA)

... Others arguments passed to the base algorithm for all subproblems.

predict.params A list of default arguments passed to the predict algorithm. (default: list())

cores Ignored because this method does not support multi-core.

seed An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Details

Nested Stacking is based on Classifier Chains transformation method to predict multi-label data. It differs from CC to predict the labels values in the training step and to regularize the output based on the labelsets available on training data.

Value

An object of class NS$\text{model}$ containing the set of fitted models, including:

- **chain** A vector with the chain order
- **labels** A vector with the label names in expected order
- **labelset** The matrix containing only labels values
- **models** A list of models named by the label names.

References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ppt, prudent, ps, rakel, rdb, rpc

Examples

```r
model <- ns(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use a specific chain with J48 classifier
mychain <- sample(rownames(toyml$labels))
model <- ns(toyml, 'J48', mychain)

# Set a specific parameter
```
Create the multi-label dataset from folds

Description

This is a simple way to use k-fold cross validation.

Usage

\[
\text{partition\_fold}(\text{kfold}, n, \text{has\_validation} = \text{FALSE})
\]

Arguments

- \text{kfold} A kFoldPartition object obtained from use of the method \text{create\_kfold\_partition}.
- \text{n} The number of the fold to separated train and test subsets.
- \text{has\_validation} Logical value that indicate if a validation dataset will be used. (Default: FALSE)

Value

A list contained train and test mldr dataset:

- \text{train} The mldr dataset with train examples, that includes all examples except those that are in test and validation samples
- \text{test} The mldr dataset with test examples, defined by the number of the fold
- \text{validation} Optionally, only if \text{has\_validation} = \text{TRUE}. The mldr dataset with validation examples

Examples

```r
folds <- create_kfold_partition(toyml, 10)

# Using the first partition
dataset <- partition\_fold(folds, 1)
names(dataset)
## [1] "train" "test"

# All iterations
for (i in 1:10) {
  dataset <- partition\_fold(folds, i)
  #dataset$train
  #dataset$test
}

# Using 3 folds validation
dataset <- partition\_fold(folds, 3, TRUE)
# dataset$train, dataset$test, #dataset$validation
```
pcut_threshold

Proportional Thresholding (PCut)

Description

Define the proportion of examples for each label will be positive. The Proportion Cut (PCut) method can be a label-wise or global method that calibrates the threshold(s) from the training data globally or per label.

Usage

pcut_threshold(prediction, ratio, probability = FALSE)

## Default S3 method:
pcut_threshold(prediction, ratio, probability = FALSE)

## S3 method for class 'mlresult'
pcut_threshold(prediction, ratio, probability = FALSE)

Arguments

- **prediction**: A matrix or mlresult.
- **ratio**: A single value between 0 and 1 or a list with ratio values contained one value per label.
- **probability**: A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: FALSE)

Value

A mlresult object.

Methods (by class)

- default: Proportional Thresholding (PCut) method for matrix
- mlresult: Proportional Thresholding (PCut) for mlresult

References


See Also

Other threshold: fixed_threshold, lcard_threshold, mcut_threshold, rcut_threshold, scut_threshold, subset_correction

Examples

prediction <- matrix(runif(16), ncol = 4)
p.cut_threshold(prediction, .45)

Description

Create a Pruned Problem Transformation model for multilabel classification.

Usage

```r
ppt(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"), p = 3,
info.loss = FALSE, ..., cores = getOption("utiml.cores", 1),
seed = getOption("utiml.seed", NA))
```

Arguments

- `mdata`: A mldr dataset used to train the binary models.
- `base.algorithm`: A string with the name of the base algorithm. (Default: `options("utiml.base.algorithm", "SVM")`)
- `p`: Number of instances to prune. All labelsets that occurs p times or less in the training data is removed. (Default: 3)
- `info.loss`: Logical value where TRUE means discard infrequent labelsets and FALSE means reintroduce infrequent labelsets via subsets. (Default: FALSE)
- `...`: Others arguments passed to the base algorithm for all subproblems
- `cores`: Not used
- `seed`: An optional integer used to set the seed. (Default: `options("utiml.seed", NA)`) 

Details

Pruned Problem Transformation (PPT) is a multi-class transformation that remove the less common classes to predict multi-label data.

Value

An object of class PPTmodel containing the set of fitted models, including:

- `labels`: A vector with the label names.
- `model`: A LP model contained only the most common labelsets.
References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbir, ebr, ecc, eps, homer, lift, lp, mbr, ns, prudent, ps, rakel, rdbr, rpc

Other Powerset: eps, lp, ps, rakel

Examples

```r
model <- ppt(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
## Change default configurations
model <- ppt(toyml, "RF", p=4, info.loss=TRUE)

## End(Not run)
```

---

**predict.BASLINEmodel**  *Predict Method for BASELINE*

Description

This function predicts values based upon a model trained by `baseline`.

Usage

```r
## S3 method for class 'BASLINEmodel'
predict(object, newdata,
    probability = getOption("ultiml.use.probs", TRUE), ...)
```

Arguments

- `object` Object of class 'BASLINEmodel'.
- `newdata` An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability` Logical indicating whether class probabilities should be returned. (Default: `getOption("ultiml.use.probs", TRUE)`)
  - not used.
- `...` not used.

Value

An object of type mlresult, based on the parameter `probability`. 
predict.BRmodel

See Also

Baseline

Examples

```r
model <- baseline(toyml)
pred <- predict(model, toyml)
```

---

**predict.BRmodel**  
*Predict Method for Binary Relevance*

**Description**

This function predicts values based upon a model trained by `br`.

**Usage**

```r
## S3 method for class 'BRmodel'
predict(object, newdata, 
  probability = getOption("utiml.use.probs", TRUE), ..., 
  cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- `object`: Object of class 'BRmodel'.
- `newdata`: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability`: Logical indicating whether class probabilities should be returned. (Default: `getOption("utiml.use.probs", TRUE)`)
- `...`: Others arguments passed to the base algorithm prediction for all subproblems.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `options("utiml.cores", 1)`)  
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `options("utiml.seed", NA)```

**Value**

An object of type mlresult, based on the parameter `probability`.

**See Also**

Binary Relevance (BR)
**predict.BRPmodel**

**Examples**

```r
model <- br(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Predict SVM scores
model <- br(toyml, "SVM")
pred <- predict(model, toyml)

# Predict SVM bipartitions running in 4 cores
pred <- predict(model, toyml, probability = FALSE, CORES = 4)

# Passing a specific parameter for SVM predict algorithm
pred <- predict(model, dataset$test, na.action = na.fail)

## End(Not run)
```

**predict.BRPmodel**  
*Predict Method for BR+ (brplus)*

**Description**

This function predicts values based upon a model trained by brplus.

**Usage**

```r
## S3 method for class 'BRPmodel'
predict(object, newdata, strategy = c("Dyn", "Stat", "Ord", "NU"), order = list(), probability = getOption("utiml.use.probs", TRUE), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- **object**  
  Object of class `BRPmodel`.

- **newdata**  
  An object containing the new input data. This must be a matrix, data.frame or a mldr object.

- **strategy**  
  The strategy prefix to determine how to estimate the values of the augmented features of unlabeled examples.
  The possible values are: 'Dyn', 'Stat', 'Ord' or 'NU'. See the description for more details. (Default: 'Dyn').

- **order**  
  The label sequence used to update the initial labels results based on the final results. This argument is used only when the `strategy` = 'Ord' (Default: list()).

- **probability**  
  Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
... Others arguments passed to the base algorithm prediction for all subproblems.

`cores` The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `options("utm1.cores", 1)`)  

`seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `options("utm1.seed", NA)`)  

**Details**  
The strategies of estimate the values of the new features are separated in two groups:

**No Update (NU)** This use the initial prediction of BR to all labels. This name is because no modification is made to the initial estimates of the augmented features during the prediction phase  

**With Update** This strategy update the initial prediction in that the final predict occurs. There are three possibilities to define the order of label sequences:  

- **Specific order** (Ord) The order is define by the user, require a new argument called `order`.  
- **Static order** (Stat) Use the frequency of single labels in the training set to define the sequence, where the least frequent labels are predicted first  
- **Dynamic order** (Dyn) Takes into account the confidence of the initial prediction for each independent single label, to define a sequence, where the labels predicted with less confidence are updated first.

**Value**  
An object of type `mlresult`, based on the parameter `probability`.

**References**  

**See Also**  
`BR+`  

**Examples**  

```r  
# Predict SVM scores  
model <- brplus(toyml, "RANDOM")  
pred <- predict(model, toyml)  

## Not run:  
# Predict SVM bipartitions and change the method to use No Update strategy  
pred <- predict(model, toyml, strategy = 'NU', probability = FALSE)  

# Predict using a random sequence to update the labels  
labels <- sample(rownames(dataset$train$labels))  
pred <- predict(model, toyml, strategy = 'Ord', order = labels)  ```
predict.CCmodel

# Passing a specific parameter for SVM predict method
pred <- predict(model, toyml, na.action = na.fail)

## End(Not run)

---

**predict.CCmodel**  
*Predict Method for Classifier Chains*

### Description

This function predicts values based upon a model trained by cc.

### Usage

```r
## S3 method for class 'CCmodel'
predict(object, newdata,  
  probability =getOption("utilm.use.probs", TRUE), ..., cores = NULL,  
  seed =getOption("utilm.seed", NA))
```

### Arguments

- **object**: Object of class 'CCmodel'.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- **probability**: Logical indicating whether class probabilities should be returned. (Default: `getOption("utilm.use.probs", TRUE)`)
- **...**: Others arguments passed to the base algorithm prediction for all subproblems.
- **cores**: Ignored because this method does not support multi-core.
- **seed**: An optional integer used to set the seed. (Default: `options("utilm.seed", NA)`)  

### Value

An object of type mlresult, based on the parameter `probability`.

### Note

The Classifier Chains prediction cannot be parallelized

### See Also

*Classifier Chains (CC)*
Examples

```r
model <- cc(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Predict SVM bipartitions
pred <- predict(model, toyml, prob = FALSE)

# Passing a specific parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)

## End(Not run)
```

---

**predict.CL Rmodel**

**Predict Method for CLR**

**Description**

This function predicts values based upon a model trained by `clr`.

**Usage**

```r
## S3 method for class 'CLRmodel'
predict(object, newdata,
    probability =getOption("utiml.use.probs", TRUE), ...,
    cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```

**Arguments**

- `object` Object of class 'CLRmodel'.
- `newdata` An object containing the new input data. This must be a matrix, data.frame or a mlr object.
- `probability` Logical indicating whether class probabilities should be returned. (Default: `getOption("utiml.use.probs", TRUE)`)  
- `...` Others arguments passed to the base algorithm prediction for all subproblems.
- `cores` The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `options("utiml.cores", 1)`)  
- `seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `options("utiml.seed", NA)`)  

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

*Binary Relevance (BR)*
**Examples**

```r
model <- clr(toyml, "RANDOM")
pred <- predict(model, toyml)
```

```r
## Not run:
```

---

**predict.CTRLmodel**  
*Predict Method for CTRL*

---

**Description**

This function predicts values based upon a model trained by `ctrl`.

**Usage**

```r
## S3 method for class 'CTRLmodel'
predict(object, newdata, vote.schema = "maj",
        probability = getOption("utiml.use.probs", TRUE), ...,
        cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- **object**  
  Object of class `CTRLmodel`.

- **newdata**  
  An object containing the new input data. This must be a matrix, data.frame or a mldr object.

- **vote.schema**  
  Define the way that ensemble must compute the predictions. The default valid options are: c("avg", "maj", "max", "min"). (Default: `maj`)  

- **probability**  
  Logical indicating whether class probabilities should be returned. (Default: `getOption("utiml.use.probs", TRUE)`)  

- **...**  
  Others arguments passed to the base algorithm prediction for all subproblems.

- **cores**  
  The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `getOption("utiml.cores", 1)`)

- **seed**  
  An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `getOption("utiml.seed", NA)`)  

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

`CTRL`
predict.DBRmodel

**Examples**

```r
model <- ctrl(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Predict SVM bipartitions running in 6 cores
pred <- predict(model, toyml, probability = FALSE, cores = 6)

# Using the Maximum vote schema
pred <- predict(model, toyml, vote.schema = 'max')

## End(Not run)
```

**Description**

This function predicts values based upon a model trained by dbr. In general this method is a restricted version of `predict.BRPmodel` using the 'NU' strategy.

**Usage**

```r
## S3 method for class 'DBRmodel'
predict(object, newdata, estimative = NULL, probability = getOption("utiml.use.probs", TRUE), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- `object`: Object of class 'DBRmodel'.
- `newdata`: An object containing the new input data. This must be a matrix, data.frame or an mlr object.
- `estimative`: A matrix containing the bipartition result of other multi-label classification algorithm or an mlresult object with the predictions.
- `probability`: Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `...`: Others arguments passed to the base algorithm prediction for all subproblems.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the \texttt{parallel} package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

**Details**

As new feature is possible to use other multi-label classifier to predict the estimate values of each label. To this use the prediction argument to inform a result of other multi-label algorithm.
**predict.EBRmodel**

**Value**

An object of type mlresult, based on the parameter probability.

**References**


**See Also**

**Dependent Binary Relevance (DBR)**

**Examples**

```r
## Not run:
# Predict SVM scores
model <- dbr(toyml)
pred <- predict(model, toyml)

# Passing a specific parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)

# Using other classifier (EBR) to make the labels estimatives
estimative <- predict(ebr(toyml), toyml)
model <- dbr(toyml, estimate.models = FALSE)
pred <- predict(model, toyml, estimative = estimative)

## End(Not run)
```

---

**predict.EBRmodel**

*Predict Method for Ensemble of Binary Relevance*

**Description**

This method predicts values based upon a model trained by `ebr`.

**Usage**

```r
## S3 method for class 'EBRmodel'
predict(object, newdata, vote.schema = "maj",
    probability =getOption("utiml.use.probs", TRUE), ..., 
    cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```
predict.ECCmodel

Predict Method for Ensemble of Classifier Chains

Arguments

- **object**: Object of class 'EBRmodel'.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- **vote.schema**: Define the way that ensemble must compute the predictions. The default valid options are: c("avg", "maj", "max", "min"). If NULL then all predictions are returned. (Default: 'maj')
- **probability**: Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- **...**: Others arguments passed to the base algorithm prediction for all subproblems.
- **cores**: The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- **seed**: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Value

An object of type mlresult, based on the parameter probability.

See Also

- Ensemble of Binary Relevance (EBR)
- Compute Multi-label Predictions

Examples

```r
## Not run:
# Predict SVM scores
model <- ebr(toyml)
pred <- predict(model, toyml)

# Predict SVM bipartitions running in 6 cores
pred <- predict(model, toyml, prob = FALSE, cores = 6)

# Return the classes with the highest score
pred <- predict(model, toyml, vote = 'max')

## End(Not run)
```

Description

This method predicts values based upon a model trained by ecc.
Usage

```r
## S3 method for class 'ECCmodel'
predict(object, newdata, vote.schema = "maj",
probability = getOption("utiml.use.probs", TRUE), ...,
cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

Arguments

- `object`: Object of class 'ECCmodel'.
- `newdata`: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `vote.schema`: Define the way that ensemble must compute the predictions. The default valid options are: c("avg", "maj", "max", "min"). If NULL then all predictions are returned. (Default: "maj")
- `probability`: Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `...`: Others arguments passed to the base algorithm prediction for all subproblems.
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the **parallel** package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Value

An object of type mldresult, based on the parameter probability.

See Also

**Ensemble of Classifier Chains (ECC)**

Examples

```r
## Not run:
# Predict SVM scores
model <- ecc(toyml)
pred <- predict(model, toyml)

# Predict SVM bipartitions running in 6 cores
pred <- predict(model, toyml, probability = FALSE, cores = 6)

# Return the classes with the highest score
pred <- predict(model, toyml, vote.schema = 'max')

## End(Not run)
```
**predict.EPSmodel**  
*Predict Method for Ensemble of Pruned Set Transformation*

**Description**

This function predicts values based upon a model trained by **eps**. Different from the others methods the probability value, is actually, the sum of all probability predictions such as it is described in the original paper.

**Usage**

```r
## S3 method for class 'EPSmodel'
predict(object, newdata, threshold = 0.5,
         probability = getOption("utiml.use.probs", TRUE), ..., 
         cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- `object` Object of class 'EPSmodel'.
- `newdata` An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `threshold` A threshold value for producing bipartitions. (Default: 0.5)
- `probability` Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `...` Others arguments passed to the base algorithm prediction for all subproblems.
- `cores` The number of cores to parallelize the prediction. Values higher than 1 require the **parallel** package. (Default: options("utiml.cores", 1))
- `seed` An optional integer used to set the seed. (Default: options("utiml.seed", NA))

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

*Ensemble of Pruned Set (EPS)*

**Examples**

```r
model <- eps(toyml, "RANDOM")
pred <- predict(model, toyml)
```
predict.HOMERmodel

**Predict Method for HOMER**

**Description**

This function predicts values based upon a model trained by *homer*.

**Usage**

```r
## S3 method for class 'HOMERmodel'
predict(object, newdata, probability =getOption("utiml.use.probs", TRUE), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- **object**
  - Object of class 'HOMERmodel'.

- **newdata**
  - An object containing the new input data. This must be a matrix, data.frame or a mldr object.

- **probability**
  - Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))

- **cores**
  - The number of cores to parallelize the prediction. Values higher than 1 require the **parallel** package. (Default: options("utiml.cores", 1))

- **seed**
  - An optional integer used to set the seed. (Default: options("utiml.seed", NA))

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

- *Hierarchy Of Multilabel classifier (HOMER)*

**Examples**

```r
model <- homer(toyml, "RANDOM")
pred <- predict(model, toyml)
```
predict.LIFTmodel  Predict Method for LIFT

Description

This function predicts values based upon a model trained by lift.

Usage

```r
## S3 method for class 'LIFTmodel'
predict(object, newdata,
    probability =getOption("utiml.use.probs", TRUE), ...,
    cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

Arguments

- `object`  Object of class 'LIFTmodel'.
- `newdata`  An object containing the new input data. This must be a matrix, data.frame or an mldr object.
- `probability`  Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `...`  Others arguments passed to the base algorithm prediction for all subproblems.
- `cores`  The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- `seed`  An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Value

An object of type mlresult, based on the parameter probability.

See Also

LIFT

Examples

```r
model <- lift(toyml, "RANDOM")
pred <- predict(model, toyml)
```
predict.LPmodel  

**Predict Method for Label Powerset**

---

**Description**

This function predicts values based upon a model trained by lp.

**Usage**

```r
## S3 method for class 'LPmodel'
predict(object, newdata,
    probability =getOption("utiml.use.probs", TRUE), ...,
    cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```

**Arguments**

- `object` Object of class 'LPmodel'.
- `newdata` An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability` Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `...` Others arguments passed to the base algorithm prediction for all subproblems.
- `cores` Not used
- `seed` An optional integer used to set the seed. (Default: options("utiml.seed", NA))

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

Label Powerset (LP)

**Examples**

```r
model <- lp(toyml, "RANDOM")
pred <- predict(model, toyml)
```
predict.MBRmodel  

**Predict Method for Meta-BR/2BR**

**Description**

This function predicts values based upon a model trained by mbr.

**Usage**

```r
## S3 method for class 'MBRmodel'
predict(object, newdata, 
    probability =getOption("uitml.use.probs", TRUE), ..., 
    cores =getOption("uitml.cores", 1), seed =getOption("uitml.seed", NA))
```

**Arguments**

- **object**: Object of class 'MBRmodel'.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- **probability**: Logical indicating whether class probabilities should be returned. (Default: `getOption("uitml.use.probs", TRUE)`)  
- **...**: Others arguments passed to the base algorithm prediction for all subproblems.
- **cores**: The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: `getOption("uitml.cores", 1))`
- **seed**: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `getOption("uitml.seed", NA))`

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

Meta-BR (MBR or 2BR)

**Examples**

```r
## Not run:
# Predict SVM scores
model <- mbr(toyml)
pred <- predict(model, toyml)

# Predict SVM bipartitions
pred <- predict(model, toyml, probability = FALSE)

# Passing a specif parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)
```
predict.NSmodel

## End(Not run)

predict.NSmodel  
*Predict Method for Nested Stacking*

**Description**

This function predicts values based upon a model trained by ns. The scores of the prediction was adapted once this method uses a correction of labelsets to predict only classes present on training data. To more information about this implementation see `subset_correction`.

**Usage**

```r
## S3 method for class 'NSmodel'
predict(object, newdata,  
        probability = getOption("utiml.use.probs", TRUE), ..., cores = NULL,  
        seed = getOption("utiml.seed", NA))
```

**Arguments**

- `object` Object of class 'NSmodel'.
- `newdata` An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability` Logical indicating whether class probabilities should be returned. (Default: `getOption("utiml.use.probs", TRUE)`)  
- `...` Others arguments passed to the base algorithm prediction for all subproblems.  
- `cores` Ignored because this method does not support multi-core.  
- `seed` An optional integer used to set the seed. (Default: `options("utiml.seed", NA)`)  

**Value**

An object of type mlresult, based on the parameter probability.

**See Also**

`Nested Stacking (NS)`

**Examples**

```r
model <- ns(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Predict SVM bipartitions  
pred <- predict(model, toyml, probability = FALSE)
```
predict.PPTmodel

Predict Method for Pruned Problem Transformation

Description

This function predicts values based upon a model trained by ppt.

Usage

## S3 method for class 'PPTmodel'
predict(object, newdata, 
  probability =getOption("utiml.use.probs", TRUE), ...,
  cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))

Arguments

object Object of class 'PPTmodel'.
newdata An object containing the new input data. This must be a matrix, data.frame or a mldr object.
probability Logical indicating whether class probabilities should be returned. (Default: 
  getOption("utiml.use.probs", TRUE))
... Others arguments passed to the base algorithm prediction for all subproblems.
cores Not used
seed An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Value

An object of type mlresult, based on the parameter probability.

See Also

Pruned Problem Transformation (PPT)

Examples

model <- ppt(toyml, "RANDOM")
pred <- predict(model, toyml)
predict.PruDentmodel  Predict Method for PruDent

Description
This function predicts values based upon a model trained by prudent.

Usage
```r
## S3 method for class 'PruDentmodel'
predict(object, newdata,
    probability =getOption("utiml.use.probs", TRUE), ...,
    cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```

Arguments
- `object`: Object of class `PruDentmodel`.
- `newdata`: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability`: Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- `cores`: The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Value
An object of type mlresult, based on the parameter probability.

See Also
- PruDent

Examples
```r
## Not run:
# Predict SVM scores
model <- prudent(toyml)
pred <- predict(model, toyml)

# Predict SVM bipartitions
pred <- predict(model, toyml, probability = FALSE)

# Passing a specific parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)
```
predict.PSmmodel

Predict Method for Pruned Set Transformation

Description
This function predicts values based upon a model trained by ps.

Usage

## S3 method for class 'PSmodel'
predict(object, newdata,
    probability = getOption("utiml.use.probs", TRUE), ...,
    cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))

Arguments

object
Object of class 'PSmodel'.

newdata
An object containing the new input data. This must be a matrix, data.frame or a
mldr object.

probability
Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))

cores
Not used

seed
An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Value
An object of type mlresult, based on the parameter probability.

See Also

Pruned Set (PS)

Examples

model <- ps(toyml, "RANDOM")
pred <- predict(model, toyml)
predict.RAkELmodel

**Predict Method for RAkEL**

**Description**

This function predicts values based upon a model trained by `rakel`.

**Usage**

```r
## S3 method for class 'RAkELmodel'
predict(object, newdata,
    probability = getOption("uti1m.l.use.probs", TRUE), ...,
    cores = getOption("uti1m.l.cores", 1), seed = getOption("uti1m.seed", NA))
```

**Arguments**

- `object`: Object of class `RAkELmodel`.
- `newdata`: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- `probability`: Logical indicating whether class probabilities should be returned. (Default: `getOption("uti1m.l.use.probs", TRUE)`)  
- `...`: Others arguments passed to the base algorithm prediction for all subproblems.
- `cores`: The number of cores to parallelize the prediction. Values higher than 1 require the `parallel` package. (Default: `getOption("uti1m.l.cores", 1)`)  
- `seed`: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: `getOption("uti1m.seed", NA)`)  

**Value**

An object of type mlresult, based on the parameter `probability`.

**See Also**

- `Random k Labelsets (RAkEL)`

**Examples**

```r
model <- rakel(toyml, "RANDOM")
pred <- predict(model, toyml)
```
**predict.RDBRmodel**  
*Predict Method for RDBR*

**Description**

This function predicts values based upon a model trained by rdbr. In general this method is a recursive version of predict.DBRmodel.

**Usage**

```r
## S3 method for class 'RDBRmodel'
predict(object, newdata, estimative = NULL, max.iterations = 5, batch.mode = FALSE, probability =getOption("utiml.use.probs", TRUE), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

**Arguments**

- **object**: Object of class 'RDBRmodel'.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
- **estimative**: A matrix containing the bipartition result of other multi-label classification algorithm or an mresult object with the predictions.
- **max.iterations**: The maximum allowed iterations of the RDBR technique. (Default: 5)
- **batch.mode**: Logical value to determine if use the batch re-estimation. If FALSE then use the stochastic re-estimation strategy. (Default: FALSE)
- **probability**: Logical indicating whether class probabilities should be returned. (Default: getOption("utiml.use.probs", TRUE))
- **...**: Others arguments passed to the base algorithm prediction for all subproblems.
- **cores**: The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- **seed**: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

**Details**

Two versions of the update strategy of the estimated labels are implemented. The batch re-estimates the labels only when a complete current label vector is available. The stochastic uses re-estimated labels as soon as they become available. This second does not support parallelize the prediction, however stabilizes earlier than batch mode.

**Value**

An object of type mlresult, based on the parameter probability.
References

See Also
Recursive Dependent Binary Relevance (RDBR)

Examples
```r
## Not run:
# Predict SVM scores
model <- rdbr(toyml)
pred <- predict(model, toyml)

# Passing a specif parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)

# Use the batch mode and increase the max number of iteration to 10
pred <- predict(model, toyml, max.iterations = 10, batch.mode = TRUE)

# Using other classifier (EBR) to made the labels estimatives
estimative <- predict(ebr(toyml), toyml, probability = FALSE)
model <- rdbr(toyml, estimate.models = FALSE)
pred <- predict(model, toyml, estimative = estimative)

## End(Not run)
```

predict.RPCmodel  
*Predict Method for RPC*

Description
This function predicts values based upon a model trained by rpc.

Usage
```r
## S3 method for class 'RPCmodel'
predict(object, newdata,  
  probability =getOption("utiml.use.probs", TRUE), ...,  
  cores =getOption("utiml.cores", 1), seed =getOption("utiml.seed", NA))
```

Arguments
- **object**: Object of class 'RPCmodel'.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a mldr object.
probability Logical indicating whether class probabilities should be returned. (Default: 
getOption("utiml.use.probs", TRUE))

... Others arguments passed to the base algorithm prediction for all subproblems.

cores The number of cores to parallelize the training. Values higher than 1 require the
parallel package. (Default: options("utiml.cores", 1))

seed An optional integer used to set the seed. This is useful when the method is run
in parallel. (Default: options("utiml.seed", NA))

Value
An object of type mlresult, based on the parameter probability.

See Also
Binary Relevance (BR)

Examples

model <- rpc(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:

---

print.BRmodel  

Print BR model

Description
Print BR model

Usage

## S3 method for class 'BRmodel'
print(x, ...)

Arguments

x The br model

... ignored
print.BRPmodel  

Print BRP model

Description

Print BRP model

Usage

```r
## S3 method for class 'BRPmodel'
print(x, ...)
```

Arguments

- `x`: The brp model
- `...`: ignored

---

print.CCmodel

Print CC model

Description

Print CC model

Usage

```r
## S3 method for class 'CCmodel'
print(x, ...)
```

Arguments

- `x`: The cc model
- `...`: ignored
print.CLRmodel

Description
Print CLR model

Usage
## S3 method for class 'CLRmodel'
print(x, ...)

Arguments
x The br model
...

print.CTRLmodel

Description
Print CTRL model

Usage
## S3 method for class 'CTRLmodel'
print(x, ...)

Arguments
x The ctrlmodel
...

 ignoring
#### Description

Print DBR model

#### Usage

```r
## S3 method for class 'DBRmodel'
print(x, ...)
```

#### Arguments

- `x`: The dbr model
- `...`: ignored

---

#### Description

Print EBR model

#### Usage

```r
## S3 method for class 'EBRmodel'
print(x, ...)
```

#### Arguments

- `x`: The ebr model
- `...`: ignored
print.ECCmodel  \hspace{1cm} \textit{Print ECC model}

\underline{Description}

Print ECC model

\underline{Usage}

\texttt{\#\# S3 method for class 'ECCmodel'}
\texttt{print(x, \ldots)}

\underline{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} The ecc model
  \item \texttt{\ldots} \hspace{1cm} ignored
\end{itemize}

print.EPSmodel  \hspace{1cm} \textit{Print EPS model}

\underline{Description}

Print EPS model

\underline{Usage}

\texttt{\#\# S3 method for class 'EPSmodel'}
\texttt{print(x, \ldots)}

\underline{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} The ps model
  \item \texttt{\ldots} \hspace{1cm} ignored
\end{itemize}
`print.kFoldPartition`  
*Print a kFoldPartition object*

---

**Description**

Print a kFoldPartition object

**Usage**

```r
## S3 method for class 'kFoldPartition'
print(x, ...)
```

**Arguments**

- `x`  
The kFoldPartition object
- `...`  
ignored

---

`print.LIFTmodel`  
*Print LIFT model*

---

**Description**

Print LIFT model

**Usage**

```r
## S3 method for class 'LIFTmodel'
print(x, ...)
```

**Arguments**

- `x`  
The lift model
- `...`  
ignored
print.LPmodel  Print LP model

Description
Print LP model

Usage

```r
## S3 method for class 'LPmodel'
print(x, ...)
```

Arguments

- `x`  
  The lp model
- `...`  
  ignored

print.majorityModel  Print Majority model

Description
Print Majority model

Usage

```r
## S3 method for class 'majorityModel'
print(x, ...)
```

Arguments

- `x`  
  The base model
- `...`  
  ignored
**print.MBRmodel**

Print MBR model

**Usage**

```r
## S3 method for class 'MBRmodel'
print(x, ...)
```

**Arguments**

- `x` The mbr model
- `...` ignored

---

**print.mlconfmat**

Print a Multi-label Confusion Matrix

**Usage**

```r
## S3 method for class 'mlconfmat'
print(x, ...)
```

**Arguments**

- `x` The mlconfmat
- `...` ignored
### print.mlresult

**Description**

Print the mlresult

**Usage**

```r
## S3 method for class 'mlresult'
print(x, ...)
```

**Arguments**

- `x` The mlresult to print
- `...` Extra parameters for print method

### print.NSmodel

**Description**

Print NS model

**Usage**

```r
## S3 method for class 'NSmodel'
print(x, ...)
```

**Arguments**

- `x` The ns model
- `...` ignored
print.PPTmodel  Print PPT model

Description
Print PPT model

Usage
## S3 method for class 'PPTmodel'
print(x, ...)

Arguments
x
The ppt model
...
ignored

print.PruDentmodel  Print PruDent model

Description
Print PruDent model

Usage
## S3 method for class 'PruDentmodel'
print(x, ...)

Arguments
x
The prudent model
...
ignored
print.PSmodel  

Print PS model

Description

Print PS model

Usage

## S3 method for class 'PSmodel'
print(x, ...)

Arguments

x  The ps model

...  ignored

print.RAkelmodel  

Print RAkEL model

Description

Print RAkEL model

Usage

## S3 method for class 'RAkELmodel'
print(x, ...)

Arguments

x  The rakel model

...  ignored
# print.randomModel

## Description
Print Random model

## Usage
```
## S3 method for class 'randomModel'
print(x, ...)
```

## Arguments
- `x`: The base model
- `...`: ignored

---

# print.RDBRmodel

## Description
Print RDBR model

## Usage
```
## S3 method for class 'RDBRmodel'
print(x, ...)
```

## Arguments
- `x`: The rdbr model
- `...`: ignored
print.RPCmodel  

**Print RPC model**

**Description**

Print RPC model

**Usage**

```r
## S3 method for class 'RPCmodel'
print(x, ...)
```

**Arguments**

- `x` The br model
- `...` ignored

---

prudent  

**PruDent classifier for multi-label Classification**

**Description**

Create a PruDent classifier to predict multi-label data. To this, two round of Binary Relevance is executed, such that, the first iteration generates new attributes to enrich the second prediction.

**Usage**

```r
prudent(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
        phi = 0, ..., cores = getOption("utiml.cores", 1),
        seed = getOption("utiml.seed", NA))
```

**Arguments**

- `mdata` A mldr dataset used to train the binary models.
- `base.algorithm` A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM")
- `phi` A value between 0 and 1 to determine the information gain. The value 0 include all labels in the second phase and the 1 none.
- `...` Others arguments passed to the base algorithm for all subproblems.
- `cores` The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
- `seed` An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))
Details

In the second phase only labels whose information gain is greater than a specific phi value is added.

Value

An object of class PrudentModel containing the set of fitted models, including:

- **labels**: A vector with the label names.
- **phi**: The value of phi parameter.
- **IG**: The matrix of Information Gain used in combination with phi parameter to define the labels used in the second step.
- **basemodel**: The BRModel used in the first iteration.
- **metamodels**: A list of models named by the label names used in the second iteration.

References


See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, ps, raket, rdbr, rpc

Examples

```r
model <- prudent(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
# Use different phi correlation with J48 classifier
model <- prudent(toyml, "J48", 0.3)

# Set a specific parameter
model <- prudent(toyml, "KNN", k=5)
```

Description

Create a Pruned Set model for multilabel classification.
Usage

ps(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"), p = 3,
strategy = c("A", "B"), b = 2, ..., cores = getOption("utiml.cores", 1),
seed = getOption("utiml.seed", NA))

Arguments

mdata A mldr dataset used to train the binary models.
base.algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
p Number of instances to prune. All labelsets that occurs p times or less in the
training data is removed. (Default: 3)
strategy The strategy (A or B) for processing infrequent labelsets. (Default: A).
b The number used by the strategy for processing infrequent labelsets.
... Others arguments passed to the base algorithm for all subproblems.
cores Not used
seed An optional integer used to set the seed. (Default: options("utiml.seed", NA))

Details

Pruned Set (PS) is a multi-class transformation that remove the less common classes to predict
multi-label data.

Value

An object of class Psmodel containing the set of fitted models, including:

labels A vector with the label names.
model A LP model contained only the most common labelsets.

References

cceedings of the New Zealand Computer Science Research Student Conference (pp. 143-150).

See Also

Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp,
mbr, ns, ppt, prudent, rakel, rdb, rpc
Other Powerset: eps, lp, ppt, rakel

Examples

model <- ps(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
## Change default configurations
rakel

Random k-labelsets for multilabel classification

Description

Create a RAkEL model for multilabel classification.

Usage

rakel(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"),
      k = 3, m = 2 * mdata$measures$num.labels, overlapping = TRUE, ...
      cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))

Arguments

mdata A mldr dataset used to train the binary models.
base.algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
k The number of labels used in each labelset. (Default: 3)
m The number of LP models. Used when overlapping is TRUE, otherwise it is ignored. (Default: 2 * length(labels))
overlapping Logical value, that defines if the method must overlapping the labelsets. If FALSE the method uses disjoint labelsets. (Default: TRUE)
... Others arguments passed to the base algorithm for all subproblems.
cores The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("utiml.cores", 1))
seed An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

Details

RAndom k labELsets is an ensemble of LP models where each classifier is trained with a small set of labels, called labelset. Two different strategies for constructing the labelsets are the disjoint and overlapping labelsets.

Value

An object of class RAkELmodel containing the set of fitted models, including:

labels A vector with the label names.
labelsets A list with the labelsets used to build the LP models.
model A list of the generated models, named by the label names.
rcut_threshold

References

See Also
Other Transformation methods: brplus, br, cc, clr, ctrl, db, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rdbr, rpc
Other Powerset: eps, lp, ppt, ps

Examples
model <- rakel(toyml, "RANDOM")
pred <- predict(model, toyml)
## Not run:
## SVM using k = 4 and m = 100
model <- rakel(toyml, "SVM", k=4, m=100)

## Random Forest using disjoint labelsets
model <- rakel(toyml, "RF", overlapping=FALSE)

## End(Not run)

rcut_threshold Rank Cut (RCut) threshold method

Description
The Rank Cut (RCut) method is an instance-wise strategy, which outputs the k labels with the highest scores for each instance at the deployment.

Usage
rcut_threshold(prediction, k, probability = FALSE)

## Default S3 method:
rcut_threshold(prediction, k, probability = FALSE)

## S3 method for class 'mlresult'
rcut_threshold(prediction, k, probability = FALSE)

Arguments
prediction A matrix or mlresult.
k The number of elements that will be positive.
probability A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: FALSE)
Value

A mlresult object.

Methods (by class)

- default: Rank Cut (RCut) threshold method for matrix
- mlresult: Rank Cut (RCut) threshold method for mlresult

References


See Also

Other threshold: fixed_threshold, lcard_threshold, mcut_threshold, pcut_threshold, scut_threshold, subset_correction

Examples

prediction <- matrix(runif(16), ncol = 4)
rcut_threshold(prediction, 2)

Description

Create a RDBR classifier to predict multi-label data. This is a recursive approach that enables the binary classifiers to discover existing label dependency by themselves. The idea of RDBR is running DBR recursively until the results stabilization of the result.

Usage

rdbr(mdata, base_algorithm = getOption("utiml.base.algorithm", "SVM"),
     estimate.models = TRUE, ..., cores = getOption("utiml.cores", 1),
     seed = getOption("utiml.seed", NA))

Arguments

mdata A mlr dataset used to train the binary models.
base_algorithm A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
estimate.models Logical value indicating whether is necessary build Binary Relevance classifier for estimate process. The default implementation use BR as estimators, however when other classifier is desirable then use the value FALSE to skip this process. (Default: TRUE).

... Others arguments passed to the base algorithm for all subproblems.

cores The number of cores to parallelize the training. Values higher than 1 require the parallel package. (Default: options("uti1.cores", 1))

seed An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("uti1.seed", NA))

Details
The train method is exactly the same of DBR the recursion is in the predict method.

Value
An object of class RDBRmodel containing the set of fitted models, including:

labels A vector with the label names.
estimation The BR model to estimate the values for the labels. Only when the estimate.models = TRUE.
models A list of final models named by the label names.

References

See Also
Dependent Binary Relevance (DBR)
Other Transformation methods: brplus, br, cc, clr, ctrl, dbr, ebr, ecc, eps, homer, lift, lp, mbr, ns, ppt, prudent, ps, rakel, rpc

Examples
model <- rdbr(toyml, "RANDOM")
pred <- predict(model, toyml)

### Not run:
# Use Random Forest as base algorithm and 4 cores
model <- rdbr(toyml, 'RF', cores = 4, seed = 123)

### End(Not run)
**remove_attributes**

*Remove attributes from the dataset*

**Description**

Remove specified attributes generating a new multi-label dataset.

**Usage**

```r
remove_attributes(mdata, attributes)
```

**Arguments**

- `mdata` (The mldr dataset to remove labels)
- `attributes` (Attributes indexes or attributes names to be removed)

**Value**

a new mldr object.

**Note**

If invalid attributes names or indexes were informed, they will be ignored.

**See Also**

Other pre process: `fill_sparse_mldata`, `normalize_mldata`, `remove_labels`, `remove_skewness_labels`, `remove_unique_attributes`, `remove_unlabeled_instances`, `replace_nominal_attributes`

**Examples**

```r
toyml1 <- remove_attributes(toyml, c("iatt8", "iatt9", "ratt10"))
toyml2 <- remove_attributes(toyml, 10)
```

---

**remove_labels**

*Remove labels from the dataset*

**Description**

Remove specified labels generating a new multi-label dataset.

**Usage**

```r
remove_labels(mdata, labels)
```
remove_skewness_labels

Arguments

mdata The mldr dataset to remove labels.
labels Label indexes or label names to be removed.

Value

a new mldr object.

Note

If invalid labels names or indexes were informed, they will be ignored.

See Also

Other pre process: fill_sparse_mldata, normalize_mldata, remove_attributes, remove_skewness_labels, remove_unique_attributes, remove_unlabeled_instances, replace_nominal_attributes

Examples

toyml1 <- remove_labels(toyml, c("y1","y5"))
toyml2 <- remove_labels(toyml, c(11, 15))

remove_skewness_labels

Remove unusual or very common labels

Description

Remove the labels that have smaller number of positive or negative examples based on a specific threshold value.

Usage

remove_skewness_labels(mdata, t = 1)

Arguments

mdata The mldr dataset to remove the skewness labels.
t Threshold value. Number of minimum examples positive and negative.

Value

a new mldr object.

See Also

Other pre process: fill_sparse_mldata, normalize_mldata, remove_attributes, remove_labels, remove_unique_attributes, remove_unlabeled_instances, replace_nominal_attributes
remove_unique_attributes

Examples

remove_skewness_labels(toyml, 20)

remove_unique_attributes

Remove unique attributes

Description

Remove the attributes that have a single value for all instances. Empty and NA values are considered different values.

Usage

remove_unique_attributes(mdata)

Arguments

mdata The mldr dataset to remove.

Value

a new mldr object.

See Also

Other pre process: fill_sparse_mldata, normalize_mldata, remove_attributes, remove_labels, remove_skewness_labels, remove_unlabeled_instances, replace_nominal_attributes

Examples

alt.toy <- toyml
alt.toy$dataset$ratt10 <- mean(alt.toy$dataset$ratt10)
new.toy <- remove_unique_attributes(alt.toy)
remove_unlabeled_instances

Remove examples without labels

Description
Remove the examples that do not have labels.

Usage
remove_unlabeled_instances(mdata)

Arguments
mdata The mldr dataset to remove the instances.

Value
a new mldr object.

See Also
Other pre process: fill_sparse_mldata, normalize_mldata, remove_attributes, remove_labels, remove_skewness_labels, remove_unique_attributes, replace_nominal_attributes

Examples
new_toy <- remove_labels(toyml, c(12,14))
remove_unlabeled_instances(new_toy)

replace_nominal_attributes

Replace nominal attributes Replace the nominal attributes by binary attributes.

Description
Replace nominal attributes Replace the nominal attributes by binary attributes.

Usage
replace_nominal_attributes(mdata, ordinal.attributes = list())
Arguments

- **mdata**: The mldr dataset to remove.
- **ordinal.attributes**: Not yet, but it will be used to specify which attributes need to be replaced.

Value

- a new mldr object.

See Also

Other pre process: `fill_sparse_mldata`, `normalize_mldata`, `remove_attributes`, `remove_labels`, `remove_skewness_labels`, `remove_unique_attributes`, `remove_unlabeled_instances`

Examples

```r
ewn <- toyml
new.column <- as.factor(sample(c("a", "b", "c"), 100, replace = TRUE))
newn$dataset$ratt0 <- new.column
head(replace_nominal_attributes(newn))
```

### Description

Create a RPC model for multilabel classification.

#### Usage

```r
rpc(mdata, base.algorithm = getOption("utiml.base.algorithm", "SVM"), ..., cores = getOption("utiml.cores", 1), seed = getOption("utiml.seed", NA))
```

#### Arguments

- **mdata**: A mldr dataset used to train the binary models.
- **base.algorithm**: A string with the name of the base algorithm. (Default: options("utiml.base.algorithm", "SVM"))
- **cores**: The number of cores to parallelize the training. Values higher than 1 require the `parallel` package. (Default: options("utiml.cores", 1))
- **seed**: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("utiml.seed", NA))

#### Details

RPC is a simple transformation method that uses pairwise classification to predict multi-label data. This is based on the one-versus-one approach to build a specific model for each label combination.
scut_threshold

Value

An object of class `RPCmodel` containing the set of fitted models, including:

- **labels**: A vector with the label names.
- **models**: A list of the generated models, named by the label names.

References


See Also

Other Transformation methods: `brplus`, `br`, `cc`, `clr`, `ctrl`, `dbr`, `ebr`, `ecc`, `eps`, `homer`, `lift`, `lp`, `mbr`, `ns`, `ppt`, `prudent`, `ps`, `rakel`, `rdbr`

Other Pairwise methods: `clr`

Examples

```r
model <- rpc(toyml, "RANDOM")
pred <- predict(model, toyml)

## Not run:
```

```

scut_threshold  SCut Score-based method

Description

This is a label-wise method that adjusts the threshold for each label to achieve a specific loss function using a validation set or cross validation.

Usage

```r
scut_threshold(prediction, expected, loss.function = NA,
               cores = getOption("utiml.cores", 1))

## Default S3 method:
scut_threshold(prediction, expected, loss.function = NA,
               cores = getOption("utiml.cores", 1))

## S3 method for class 'mlresult'
scut_threshold(prediction, expected, loss.function = NA,
               cores = getOption("utiml.cores", 1))

```
scut_threshold

Arguments

- **prediction**: A matrix or mlresult.
- **expected**: The expected labels for the prediction. May be a matrix with the label values or a mldr object.
- **loss.function**: A loss function to be optimized. If you want to use your own error function see the notes and example. (Default: Mean Squared Error)
- **cores**: The number of cores to parallelize the computation. Values higher than 1 require the `parallel` package. (Default: `options("utiml.cores", 1)``

Details

Different from the others threshold methods instead of return the bipartition results, it returns the threshold values for each label.

Value

A numeric vector with the threshold values for each label

Methods (by class)

- **default**: Default scut_threshold
- **mlresult**: Mlresult scut_threshold

Note

The loss function is a R method that receive two vectors, the expected values of the label and the predicted values, respectively. Positive values are represented by the 1 and the negative by the 0.

References


See Also

Other threshold: `fixed_threshold`, `lcard_threshold`, `mcut_threshold`, `pcut_threshold`, `rcut_threshold`, `subset_correction`

Examples

```r
names <- list(1:10, c("a", "b", "c"))
prediction <- matrix(runif(30), ncol = 3, dimnames = names)
classes <- matrix(sample(0:1, 30, rep = TRUE), ncol = 3, dimnames = names)
thresholds <- scut_threshold(prediction, classes)
fixed_threshold(prediction, thresholds)
```
## Not run:

```r
# Penalizes only FP predictions
mylossfunc <- function (real, predicted) {
  mean(predicted - real * predicted)
}
prediction <- predict(br(toyml, "RANDOM"), toyml)
scut_threshold(prediction, toyml, loss.function = mylossfunc, cores = 5)

## End(Not run)
```

---

### subset_correction

#### Subset Correction of a predicted result

**Description**

This method restrict a multi-label learner to predict only label combinations whose existence is present in the (training) data. To this all labelsets that are predicted but are not found on training data is replaced by the most similar labelset.

**Usage**

```r
subset_correction(mlresult, train_y, probability = FALSE)
```

**Arguments**

- `mlresult`: An object of mlresult that contain the scores and bipartition values.
- `train_y`: A matrix/data.frame with all labels values of the training dataset or a mldr train dataset.
- `probability`: A logical value. If `TRUE` the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: `FALSE`)

**Details**

If the most similar is not unique, those label combinations with higher frequency in the training data are preferred. The Hamming loss distance is used to determine the difference between the labelsets.

**Value**

A new mlresult where all results are present in the training labelsets.

**Note**

The original paper describes a method to create only bipartitions result, but we adapted the method to change the scores. Based on the `base.threshold` value the scores higher than the threshold value, but must be lower are changed to respect this restriction. If `NULL` this correction will be ignored.
References


See Also

Other threshold: fixed_threshold, lcard_threshold, mcut_threshold, pcut_threshold, rcut_threshold, scut_threshold

Examples

prediction <- predict(br(toyml, "RANDOM"), toyml)
subset_correction(prediction, toyml)

toyml

Toy multi-label dataset.

Description

A toy multi-label dataset is a synthetic dataset generated by the tool http://sites.labic.icmc.usp.br/mldatagen/ using the Hyperspheres strategy. Its purpose is to be used for small tests and examples.

Usage

toyml
Format

A mldr object with 100 instances, 10 features and 5 labels:

- **att1**: Relevant numeric attribute between (-1 and 1)
- **att2**: Relevant numeric attribute between (-1 and 1)
- **att3**: Relevant numeric attribute between (-1 and 1)
- **att4**: Relevant numeric attribute between (-1 and 1)
- **att5**: Relevant numeric attribute between (-1 and 1)
- **att6**: Relevant numeric attribute between (-1 and 1)
- **att7**: Relevant numeric attribute between (-1 and 1)
- **iatt8**: Irrelevant numeric attribute between (-1 and 1)
- **iatt9**: Irrelevant numeric attribute between (-1 and 1)
- **ratt10**: Redundant numeric attribute between (-1 and 1)

- **y1**: Label 'y1' - Frequency: 0.17
- **y2**: Label 'y2' - Frequency: 0.78
- **y3**: Label 'y3' - Frequency: 0.19
- **y4**: Label 'y4' - Frequency: 0.69
- **y5**: Label 'y5' - Frequency: 0.17

Details

General Information

- Cardinality: 2
- Density: 0.4
- Distinct multi-labels: 18
- Number of single labelsets: 5
- Max frequency: 23

Source

Generated by [http://sites.labic.icmc.usp.br/mldatagen/](http://sites.labic.icmc.usp.br/mldatagen/) Configuration:

- Strategy: Hyperspheres
- Relevant Features: 7
- Irrelevant Features: 2
- Redundant Features: 1
- Number of Labels (q): 5
- Number of Instances: 100
- Noise (from 0 to 1): 0.05
- Maximum Radius/Half-Edge of the Hyperspheres/Hypercubes: 0.8
- Minimum Radius/Half-Edge of the Hyperspheres/Hypercubes: ((q/10)+1)/q
utiml

Description

The utiml package is a framework to support multi-label processing, like Mulan on Weka. The main advantage is because it is in R, that in other others, it is simple to use and extend.

Details

Currently, the main methods supported are:

1. **Classification methods**: Binary Relevance (BR), BR+, Classifier Chains, ConTRolled Label correlation exploitation (CTRL), Dependent Binary Relevance (DBR), Ensemble of Binary Relevance (EBR), Ensemble of Classifier Chains (ECC), Meta-Binary Relevance (MBR or 2BR), Nested Stacking (NS), Pruned and Confident Stacking Approach (PASA), Recursive Dependent Binary Relevance (RDBR)

2. **Evaluation methods**: Confusion Matrix, Evaluate, Supported measures

3. **Pre-process utilities**: Fill sparse data, Normalize data, Remove attributes, Remove labels, Remove skewness labels, Remove unique attributes, Remove unlabeled instances, Replace nominal attributes

4. **Sampling methods**: Create holdout partitions, Create k-fold partitions, Create random subset, Create subset, Partition fold

5. **Threshold methods**: Fixed threshold, MCUT, PCUT, RCUT, SCUT, Subset correction

However, there are other utilities methods not previously cited as as.bipartition, as.mlresult, as.ranking, multilabel_prediction, etc. More details and examples are available on utiml repository.

Notes

We use the mldr package, to manipulate multi-label data. See its documentation to more information about handle multi-label dataset.

Author(s)

- Adriano Rivolli <rivolli@utfpr.edu.br>

This package is a result of my PhD at Institute of Mathematics and Computer Sciences (ICMC) at the University of Sao Paulo, Brazil.

PhD advisor: Andre C. P. L. F. de Carvalho
utiml_all_measures_names

**Description**

Return the tree with the measure names

**Usage**

```r
utiml_all_measures_names()
```

**Value**

- `list`

---

**utiml_compute_ensemble**

*Compute binary predictions*

**Description**

Compute binary predictions

**Usage**

```r
utiml_compute_ensemble(bipartitions, probabilities, vote.methods, rnames)
```

**Arguments**

- `bipartitions`: A matrix with bipartitions values.
- `probabilities`: A matrix with probabilities values.
- `vote.methods`: The vote schema method.
- `rnames`: The row names.

**Value**

A binary.prediction object
utiml_ensemble_average

Average vote combination for a single-label prediction

Description

Compute the prediction for a single-label using the average votes schema. The probabilities result is computed using the averaged values.

Usage

utiml_ensemble_average(bipartition, probability)

Arguments

bipartition A matrix with all bipartition predictions for a single label. The column are the predictions and the rows the examples.
probability A matrix with all probability predictions for a single label. The column are the predictions and the rows the examples.

Value

A list with two values "bipartition" and "probability".

utiml_ensemble_check_voteschema

Verify if a schema vote name is valid

Description

Verify if a schema vote name is valid

Usage

utiml_ensemble_check_voteschema(vote.schema, accept.null = TRUE)

Arguments

vote.schema The name of schema vote
accept.null Logical value determine if the vote.schema = NULL is also valid. (Default: TRUE)

Value

TRUE or throw an error message otherwise
utiml_ensemble_majority

Majority vote combination for single-label prediction

Description

Compute the single-label prediction using the majority votes schema. The probabilities result is computed using only the majority instances. In others words, if a example is predicted as positive, only the positive confidences are used to compute the averaged value.

Usage

utiml_ensemble_majority(bipartition, probability)

Arguments

bipartition A matrix with all bipartition predictions for a single label. The column are the predictions and the rows the examples.
probability A matrix with all probability predictions for a single label The column are the predictions and the rows the examples.

Value

A list with two values "bipartition" and "probability".

utiml_ensemble_maximum

Maximum vote combination for single-label prediction

Description

Compute the single-label prediction using the maximum votes schema. The probabilities result is computed using the maximum value.

Usage

utiml_ensemble_maximum(bipartition, probability)

Arguments

bipartition A matrix with all bipartition predictions for a single label. The column are the predictions and the rows the examples.
probability A matrix with all probability predictions for a single label The column are the predictions and the rows the examples.
**utiml_ensemble_method**

**Value**

A list with two values "bipartition" and "probability".

**Description**

Define the method name related with the vote schema

**Usage**

`utiml_ensemble_method(vote.schema)`

**Arguments**

- `vote.schema` Define the way that ensemble must compute the predictions.

**Value**

The method name that will compute the votes

---

**utiml_ensemble_minimum**

*Minimum vote combination for single-label prediction*

**Description**

Compute the single-label prediction using the minimum votes schema. The probabilities result is computed using the minimum value.

**Usage**

`utiml_ensemble_minimum(bipartition, probability)`

**Arguments**

- `bipartition` A matrix with all bipartition predictions for a single label. The column are the predictions and the rows the examples.
- `probability` A matrix with all probability predictions for a single label. The column are the predictions and the rows the examples.

**Value**

A list with two values "bipartition" and "probability".
utiml_ifelse  Conditional value selection

Description

Conditional value selection

Usage

utiml_ifelse(test, yes, no)

Arguments

test an object which can be coerced to logical mode.
yes object that will be returned when the test value is true.
no object that will be returned when the test value is false

Value

The respective value yes or no based on test value. This is an alternative way to use a single logical value for avoid the real if/else for choice lists, matrices and other composed data.

Examples

## Not run:
utiml_ifelse(TRUE, dataframe1, dataframe2) ## dataframe1
utiml_ifelse(length(my.list) > 10, my.list[1:10], my.list)

## End(Not run)

utiml_is_equal_sets  Define if two sets are equals independently of the order of the elements

Description

Define if two sets are equals independently of the order of the elements

Usage

utiml_is_equal_sets(a, b)

Arguments

a A list
b Other list
utiml_iterative_split

Value

Logical value where TRUE the sets are equals and FALSE otherwise.

Examples

```r
## Not run:
utiml_is_equal_sets(c(1, 2, 3), c(3, 2, 1))
## TRUE

utiml_is_equal_sets(c(1, 2, 3), c(1, 2, 3, 4))
## FALSE

## End(Not run)
```

### utiml_iterative_split: Internal Iterative Stratification

Description

Create the indexes using the Iterative Stratification algorithm.

Usage

```r
utiml_iterative_split(mdata, r)
```

Arguments

- `mdata`: A mldr dataset.
- `r`: Desired proportion of examples in each subset $r_1, \ldots r_k$.

Value

A list with $k$ disjoint indexes subsets $S_1, \ldots S_k$.

References


Examples

```r
## Not run:
# Create 3 partitions for train, validation and test
indexes <- utiml_iterative_split(emotions, c(0.6,0.1,0.3))

# Create a stratified 10-fold
indexes <- utiml_iterative_split(emotions, rep(0.1,10))

## End(Not run)
```
utiml_labels_correlation

*Phi Correlation Coefficient*

**Description**

Calculate all labels phi correlation coefficient. This is a specialized version of the Pearson product moment correlation coefficient for categorical variables with two values, also called dichotomous variables. This is also called of Pearson product moment Correlation Coefficient (PCC)

**Usage**

`utiml_labels_correlation(mdata)`

**Arguments**

- `mdata`: A mlr multi-label dataset

**Value**

A matrix with all labels correlation coefficient. The rows and columns have the labels and each value are the correlation between the labels. The main diagonal have the 1 value that represents the correlation of a label with itself.

**References**


**See Also**

`metaMbr` (MBR or 2BR)

**Examples**

```r
## result <- utiml_labels_correlation(toyml)

# Get the phi coefficient between the labels 'y1' and 'y2'
## result['y1', 'y2']

# Get all coefficients of a specific label
## result[4, -4]
```
utiml_labels_IG  

**utiml_labels_IG**  
*Calculate the Information Gain for each pair of labels*

**Description**

Calculate the Information Gain for each pair of labels

**Usage**

```r
utiml_labels_IG(mdata)
```

**Arguments**

- **mdata**: A mldr dataset containing the label information.

**Value**

A matrix where the rows and columns represents the labels.

**References**


---

utiml_lapply  

**utiml_lapply**  
*Select the suitable method lapply or mclapply*

**Description**

Select the suitable method lapply or mclapply

**Usage**

```r
utiml_lapply(mylist, myfnc, utiml.cores, utiml.seed = NA, ...)
```

**Arguments**

- **mylist**: a list to iterate.
- **myfnc**: The function to be applied to each element of the mylist.
- **utiml.cores**: The number of cores to use. If 1 use lapply otherwise use mclapply.
- **utiml.seed**: A numeric value to set a seed to execute in parallel mode.
- ... Extra arguments to myfnc.

**Value**

A list with the results of the specified method.
utiml_measure_accuracy

MULTILABEL MEASURES

Multi-label Accuracy Measure

Description

MULTILABEL MEASURES

Multi-label Accuracy Measure

Usage

`utiml_measure_accuracy(mlconfmat, ...)`

Arguments

- `mlconfmat`: Confusion matrix
- `...`: ignored

References


utiml_measure_average_precision

Multi-label Average Precision Measure

Description

Multi-label Average Precision Measure

Usage

`utiml_measure_average_precision(mlconfmat, ...)`

Arguments

- `mlconfmat`: Confusion matrix
- `...`: ignored

References

**BINARY MEASURES**

Compute the binary accuracy

**Description**

Compute the binary accuracy

**Usage**

`utiml_measure_binary_accuracy(TP, FP, TN, FN)`

**Arguments**

- **TP**: The number of True Positive values
- **FP**: The number of False Positive values
- **TN**: The number of True Negative values
- **FN**: The number of False Negative values

**Value**

Accuracy value between 0 and 1

---

**utiml_measure_binary_auc**

Compute the binary AUC

**Description**

Compute the binary AUC

**Usage**

`utiml_measure_binary_auc(scores, labels)`

**Arguments**

- **scores**: The probability/score from a single label
- **labels**: The expected label predictions

**Value**

AUC value between 0 and 1
utiml_measure_binary_balacc

*Compute the binary balanced accuracy*

**Description**

Compute the binary balanced accuracy

**Usage**

```plaintext
utiml_measure_binary_balacc(TP, FP, TN, FN)
```

**Arguments**

- **TP**: The number of True Positive values
- **FP**: The number of False Positive values
- **TN**: The number of True Negative values
- **FN**: The number of False Negative values

**Value**

Balanced accuracy value between 0 and 1

---

utiml_measure_binary_f1

*Compute the binary F1 measure*

**Description**

Compute the binary F1 measure

**Usage**

```plaintext
utiml_measure_binary_f1(TP, FP, TN, FN)
```

**Arguments**

- **TP**: The number of True Positive values
- **FP**: The number of False Positive values
- **TN**: The number of True Negative values
- **FN**: The number of False Negative values

**Value**

F1 measure value between 0 and 1
**utiml_measure_binary_precision**

*Compute the binary precision*

**Description**
Compute the binary precision

**Usage**

`utiml_measure_binary_precision(TP, FP, TN, FN)`

**Arguments**

- **TP**  
The number of True Positive values
- **FP**  
The number of False Positive values
- **TN**  
The number of True Negative values
- **FN**  
The number of False Negative values

**Value**

Precision value between 0 and 1

---

**utiml_measure_binary_recall**

*Compute the binary recall*

**Description**
Compute the binary recall

**Usage**

`utiml_measure_binary_recall(TP, FP, TN, FN)`

**Arguments**

- **TP**  
The number of True Positive values
- **FP**  
The number of False Positive values
- **TN**  
The number of True Negative values
- **FN**  
The number of False Negative values

**Value**

Recall value between 0 and 1
### utiml_measure_coverage

*Multi-label Coverage Measure*

**Description**

Multi-label Coverage Measure

**Usage**

```python
utiml_measure_coverage(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


---

### utiml_measure_f1

*Multi-label F1 Measure*

**Description**

Multi-label F1 Measure

**Usage**

```python
utiml_measure_f1(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**

**utiml_measure_hamming_loss**

*Multi-label Hamming Loss Measure*

**Description**

Multi-label Hamming Loss Measure

**Usage**

```
utiml_measure_hamming_loss(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


---

**utiml_measure_is_error**

*Multi-label Is Error Measure*

**Description**

Multi-label Is Error Measure

**Usage**

```
utiml_measure_is_error(mlconfmat, ranking, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `ranking`: The expected matrix ranking
- `...`: ignored

**References**

Multi-label Macro-Accuracy Measure

**Description**

Multi-label Macro-Accuracy Measure

**Usage**

```r
utiml_measure_macro_accuracy(mlconfmat, ...)```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


Multi-label Macro-AUC Measure

**Description**

Multi-label Macro-AUC Measure

**Usage**

```r
utiml_measure_macro_AUC(mlconfmat, ...)```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**

**utiml_measure_macro_f1**

*Multi-label Macro-F1 Measure*

**Description**

Multi-label Macro-F1 Measure

**Usage**

```python
utiml_measure_macro_f1(mlconfmat, ...)
```

**Arguments**

- `mlconfmat` : Confusion matrix
- `...` : ignored

**References**


---

**utiml_measure_macro_precision**

*Multi-label Macro-Precision Measure*

**Description**

Multi-label Macro-Precision Measure

**Usage**

```python
utiml_measure_macro_precision(mlconfmat, ...)
```

**Arguments**

- `mlconfmat` : Confusion matrix
- `...` : ignored

**References**

**utiml_measure_macro_recall**

*Multi-label Macro-Recall Measure*

**Description**

Multi-label Macro-Recall Measure

**Usage**

```python
utiml_measure_macro_recall(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


---

**utiml_measure_margin_loss**

*Multi-label Margin Loss Measure*

**Description**

Multi-label Margin Loss Measure

**Usage**

```python
utiml_measure_margin_loss(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**

**utiml_measure_micro_accuracy**

*Multi-label Micro-Accuracy Measure*

**Description**

Multi-label Micro-Accuracy Measure

**Usage**

```
utiml_measure_micro_accuracy(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


---

**utiml_measure_micro_AUC**

*Multi-label Macro-AUC Measure*

**Description**

Multi-label Macro-AUC Measure

**Usage**

```
utiml_measure_micro_AUC(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**

**utiml_measure_micro_f1**

*Multi-label Micro-F1 Measure*

**Description**

Multi-label Micro-F1 Measure

**Usage**

```python
utiml_measure_micro_f1(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**


---

**utiml_measure_micro_precision**

*Multi-label Micro-Precision Measure*

**Description**

Multi-label Micro-Precision Measure

**Usage**

```python
utiml_measure_micro_precision(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`: Confusion matrix
- `...`: ignored

**References**

**Multi-label Micro-Recall Measure**

**Description**

Multi-label Micro-Recall Measure

**Usage**

```r
utiml_measure_micro_recall(mlconfmat, ...)
```

**Arguments**

- `mlconfmat`  
  Confusion matrix
- `...`  
  ignored

**References**


---

**Return the name of measures**

**Description**

Return the name of measures

**Usage**

```r
utiml_measure_names(measures = c("all"))
```

**Arguments**

- `measures`  
  The group of measures (Default: "all").

**Value**

array of character contained the measures names.
Examples

```r
## Not run:
utiml_measure_names()
utiml_measure_names("bipartition")
utiml_measure_names(c("micro-based", "macro-based"))
## End(Not run)
```

---

**utiml_measure_one_error**

*Multi-label One Error Measure*

Description

Multi-label One Error Measure

Usage

```r
utiml_measure_one_error(mlconfmat, ...)
```

Arguments

- `mlconfmat` Confusion matrix
- `...` ignored

References


---

**utiml_measure_precision**

*Multi-label Precision Measure*

Description

Multi-label Precision Measure

Usage

```r
utiml_measure_precision(mlconfmat, ...)
```

Arguments

- `mlconfmat` Confusion matrix
- `...` ignored
References


utiml_measure_ranking_error

*Multi-label Ranking Error Measure*

Description

Multi-label Ranking Error Measure

Usage

\[
\text{utiml\_measure\_ranking\_error}(\text{mlconfmat}, \text{ranking}, \ldots)
\]

Arguments

- \text{mlconfmat} \quad \text{Confusion matrix}
- \text{ranking} \quad \text{A matrix ranking}
- \ldots \quad \text{ignored}

References


utiml_measure_ranking_loss

*Multi-label Hamming Loss Measure*

Description

Multi-label Hamming Loss Measure

Usage

\[
\text{utiml\_measure\_ranking\_loss}(\text{mlconfmat}, \ldots)
\]

Arguments

- \text{mlconfmat} \quad \text{Confusion matrix}
- \ldots \quad \text{ignored}
References


utiml_measure_recall  
Multi-label Recall Measure

Description

Multi-label Recall Measure

Usage

```
utiml_measure_recall(mlconfmat, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlconfmat</td>
<td>Confusion matrix</td>
</tr>
<tr>
<td>...</td>
<td>ignored</td>
</tr>
</tbody>
</table>

References


utiml_measure_subset_accuracy  
Multi-label Subset Accuracy Measure

Description

Multi-label Subset Accuracy Measure

Usage

```
utiml_measure_subset_accuracy(mlconfmat, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlconfmat</td>
<td>Confusion matrix</td>
</tr>
<tr>
<td>...</td>
<td>ignored</td>
</tr>
</tbody>
</table>

References

utiml_newdata

Return the newdata to a data.frame or matrix

Description

Return the newdata to a data.frame or matrix

Usage

utiml_newdata(newdata)

## Default S3 method:
utiml_newdata(newdata)

## S3 method for class 'mldr'
utiml_newdata(newdata)

Arguments

newdata The data.frame or mldr data

Value

A dataframe or matrix containing only dataset

Methods (by class)

- default: Return the data in the original format
- mldr: Return the dataset from the mldr dataset

Examples

## Not run:
test <- emotions$dataset[,emotions$attributesIndexes]
all(test == utiml_newdata(emotions)) # TRUE
all(test == utiml_newdata(test)) # TRUE

## End(Not run)
utiml_normalize  
*Internal normalize data function*

**Description**

Internal normalize data function

**Usage**

```
utiml_normalize(data, max.val = NULL, min.val = NULL)
```

**Arguments**

- `data`  
a set of numbers.
- `max.val`  
The maximum value to normalize. If NULL use the max value present in the data. (default: NULL)
- `min.val`  
The minimum value to normalize. If NULL use the min value present in the data (default: NULL)

**Value**

The normalized data

**Examples**

```r
## Not run:
utiml_normalize(c(1,2,3,4,5))
#--> 0 0.25 0.5 0.75 1

utiml_normalize(c(1,2,3,4,5), 10, 0)
#--> 0.1 0.2 0.3 0.4 0.5

## End(Not run)
```

---

**utiml_predict_binary_ensemble**  
*Predict binary predictions*

**Description**

Is very similar from utiml_compute_ensemble but differs from arguments

**Usage**

```
utiml_predict_binary_ensemble(predictions, vote.schema)
```
Arguments

predictions A list of binary predictions.
vote.schema The name of vote schema.

Value
A binary.prediction object

---

**utiml_preserve_seed**

*Preserve current seed*

**Description**
Preserve current seed

**Usage**

`utiml_preserve_seed()`

---

**utiml_random_split**

*Random split of a dataset*

**Description**
Random split of a dataset

**Usage**

```r
utiml_random_split(mdata, r)
```

**Arguments**

- `mdata` A mldr dataset.
- `r` Desired proportion of examples in each subset r_1, ... r_k.

**Value**
A list with k disjoint indexes subsets S_1, . . . S_k.

**Examples**

```r
## Not run:
utiml_random_split(emotions, c(0.6, 0.2, 0.2))

## End(Not run)
```
utiml_rename  Rename the list using the names values or its own content

Description
Rename the list using the names values or its own content

Usage
`utiml_rename(X, names = NULL)`

Arguments
- **X**: A list
- **names**: The list names, If empty the content of X is used

Value
A list with the new names

Examples
```r
utiml_rename(c("a", "b", "c"))
## c(a="a", b="b", c="c")

utiml_rename(c(1, 2, 3), c("a", "b", "c"))
## c(a=1, b=2, c=3)
```

utiml_restore_seed  Restore the current seed

Description
Restore the current seed

Usage
`utiml_restore_seed()`
Labelsets Stratification Create the indexes using the Labelsets Stratification approach.

Description

Labelsets Stratification Create the indexes using the Labelsets Stratification approach.

Usage

```r
utiml_stratified_split(mdata, r)
```

Arguments

- `mdata`: A mlr dataset
- `r`: Desired proportion of examples in each subset, $r_1, \ldots, r_k$

Value

A list with $k$ disjoint indexes subsets $S_1, \ldots, S_k$

References


Examples

```r
## Not run:
# Create 3 partitions for train, validation and test
indexes <- utiml_stratified_split(emotions, c(0.6,0.1,0.3))

# Create a stratified 10-fold
indexes <- utiml_stratified_split(emotions, rep(0.1,10))

## End(Not run)
```
utiml_validate_splitmethod

Return the name of split method and validate if it is valid

Description

Return the name of split method and validate if it is valid

Usage

utiml_validate_splitmethod(method)

Arguments

method

The method name

Value

The correct name of split method

.[mlresult]

Filter a Multi-Label Result

Description

If column filter is performed, then the result will be a matrix. Otherwise, the result will be a mlresult.

Usage

## S3 method for class 'mlresult'
mlresult[rowFilter = T, colFilter, ...]

Arguments

mlresult

A mlresult object

rowFilter

A list of rows to filter

colFilter

A list of columns to filter

...  
Extra parameters to be used as the filter

Value

mlresult or matrix. If column filter is performed, then the result will be a matrix. Otherwise, the result will be a mlresult.
Index

*Topic datasets
  toyml, 105
  +.mlconfmat, 5
  [+.mlresult, 136

as.bipartition, 6, 107
as.matrix.mlresult, 6
as.mlresult, 7, 107
as.probability, 8
as.ranking, 8, 107

Baseline, 56
baseline, 9, 55
br, 10, 12, 13, 15, 22, 24, 25, 27, 28, 32, 35,
  36, 38, 51, 55, 56, 91, 92, 94, 96, 102
BR+, 58, 107
bplus, 11, 11, 13, 15, 22, 24, 25, 27, 28, 32,
  35, 36, 38, 51, 55, 91, 92, 94, 96, 102
cc, 11, 12, 15, 22, 24, 25, 27, 28, 32, 35,
  36, 38, 51, 55, 91, 92, 94, 96, 102
clr, 11–13, 14, 22, 24, 25, 27, 28, 32, 35, 36,
  38, 51, 55, 60, 91, 92, 94, 96, 102
compute_multilabel_predictions, 15
create_holdout_partition, 16, 19–21
create_kfold_partition, 17, 18, 20, 21, 52
create_random_subset, 17, 19, 19, 21
create_subset, 17, 19, 20, 20
CTRL, 61
ctrl, 11–13, 15, 21, 24, 25, 27, 28, 32, 35, 36,
  38, 51, 55, 61, 91, 92, 94, 96, 102
dbr, 11–13, 15, 22, 23, 25, 27, 28, 32, 35, 36,
  38, 51, 55, 91, 92, 94, 96, 102
ebr, 11–13, 15, 22, 24, 24, 27, 28, 32, 35, 36,
  38, 51, 55, 63, 91, 92, 94, 96, 102
ecc, 11–13, 15, 22, 24, 25, 26, 28, 32, 35, 36,
  38, 51, 55, 64, 91, 92, 94, 96, 102
eps, 11–13, 15, 22, 24, 25, 27, 27, 32, 35, 36,
  38, 51, 55, 66, 91, 92, 94, 96, 102

Evaluate, 107
fill_sparse_mldata, 29, 50, 97–101
fixed_threshold, 29, 34, 39, 54, 95, 103, 105
homer, 11–13, 15, 22, 24, 25, 27, 28, 31, 35,
  36, 38, 51, 55, 67, 91, 92, 94, 96, 102
How to create the datasets from folds, 19
is.bipartition, 32
is.probability, 33
lcard_threshold, 30, 33, 39, 54, 95, 103, 105
LIFT, 68
lift, 11–13, 15, 22, 24, 25, 27, 28, 32, 34, 36,
  38, 51, 55, 68, 91, 92, 94, 96, 102
lp, 11–13, 15, 22, 24, 25, 27, 28, 32, 35, 36,
  38, 51, 55, 69, 91, 92, 94, 96, 102
mbr, 11–13, 15, 22, 24, 25, 27, 28, 32, 35, 36,
  36, 51, 55, 91, 92, 94, 96, 102
MCUT, 107
mcut_threshold, 30, 34, 38, 54, 95, 103, 105
merge_mlconfmat, 39
mldata, 40
mldr, 107
mpredict, 40, 43
mltrain, 40, 43
multilabel_confusion_matrix, 45
multilabel_evaluate, 47
multilabel_measures, 48
multilabel_prediction, 49, 107
normalize_mldata, 29, 50, 97–101
ns, 11–13, 15, 22, 24, 25, 27, 28, 32, 35, 36,
  38, 50, 55, 91, 92, 94, 96, 102
partition_fold, 52
PCUT, 107
pcut_threshold, 30, 34, 39, 53, 95, 103, 105
utiml_lapply, 115
utiml_measure_accuracy, 116
utiml_measure_average_precision, 116
utiml_measure_binary_accuracy, 117
utiml_measure_binary_AUC, 117
utiml_measure_binary_balacc, 118
utiml_measure_binary_f1, 118
utiml_measure_binary_precision, 119
utiml_measure_binary_recall, 119
utiml_measure_coverage, 120
utiml_measure_f1, 120
utiml_measure_hamming_loss, 121
utiml_measure_is_error, 121
utiml_measure_macro_accuracy, 122
utiml_measure_macro_AUC, 122
utiml_measure_macro_f1, 123
utiml_measure_macro_precision, 123
utiml_measure_macro_recall, 124
utiml_measure_margin_loss, 124
utiml_measure_micro_accuracy, 125
utiml_measure_micro_AUC, 125
utiml_measure_micro_f1, 126
utiml_measure_micro_precision, 126
utiml_measure_micro_recall, 127
utiml_measure_names, 127
utiml_measure_one_error, 128
utiml_measure_precision, 128
utiml_measure_ranking_error, 129
utiml_measure_ranking_loss, 129
utiml_measure_recall, 130
utiml_measure_subset_accuracy, 130
utiml_newdata, 131
utiml_normalize, 132
utiml_predict_binary_ensemble, 132
utiml_predict_ensemble (compute_multilabel_predictions), 15
utiml_preserve_seed, 133
utiml_random_split, 133
utiml_rename, 134
utiml_restore_seed, 134
utiml_stratified_split, 135
utiml_validate_splitmethod, 136