Package ‘utiml’

February 7, 2020

Type Package

Title Utilities for Multi-Label Learning

Version 0.1.6

Date 2020-02-06


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

Depends R (>= 3.0.0), mldr(>= 0.4.0), parallel, ROCR

Imports stats, utils, methods

Suggests C50, e1071, FSelector, infotheo, kknn, knitr, 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 7.0.2

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author Adriano Rivolli [aut, cre]

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

Repository CRAN

Date/Publication 2020-02-07 06:30:02 UTC
R topics documented:

+ mlconfmat .................................................. 4
as.bipartition .............................................. 5
as.matrix.mlconfmat ........................................ 5
as.matrix.mlresult .......................................... 6
as.mlresult .................................................. 6
as.probability ............................................... 7
as.ranking .................................................... 8
baseline ....................................................... 8
br ............................................................. 9
brplus ......................................................... 11
cc ........................................................... 12
clr ........................................................... 14
compute_multilabel_predictions ............................. 15
create_holdout_partition .................................... 16
create_kfold_partition ....................................... 18
create_random_subset ....................................... 20
create_subset ............................................... 21
ctrl .......................................................... 22
cv ........................................................... 24
dbr ........................................................... 25
ebr ........................................................... 26
ecc ........................................................... 28
eps ........................................................... 30
esl ........................................................... 32
fill_sparse_mldata ........................................... 33
fixed_threshold .............................................. 34
foodtruck ..................................................... 35
homer ........................................................ 36
is.bipartition ............................................... 38
is.probability ............................................... 38
lcard_threshold ............................................. 39
lift ........................................................... 40
lp ............................................................ 41
mbr ........................................................... 42
mcut_threshold .............................................. 44
merge_mlconfmat .............................................. 45
mldata ........................................................ 45
mlknn ........................................................ 46
mlpredict ...................................................... 47
mltrain ........................................................ 48
multilabel_confusion_matrix ................................ 50
multilabel_evaluate .......................................... 51
multilabel_measures ......................................... 53
multilabel_prediction ....................................... 54
normalize_mldata ............................................ 55
ns ............................................................ 55
R topics documented:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_fold</td>
<td>57</td>
</tr>
<tr>
<td>pcut_threshold</td>
<td>58</td>
</tr>
<tr>
<td>ppt</td>
<td>59</td>
</tr>
<tr>
<td>predict.BASELINEmodel</td>
<td>60</td>
</tr>
<tr>
<td>predict.BRmodel</td>
<td>61</td>
</tr>
<tr>
<td>predict.BRPmodel</td>
<td>62</td>
</tr>
<tr>
<td>predict.CCmodel</td>
<td>64</td>
</tr>
<tr>
<td>predict.CLRmodel</td>
<td>65</td>
</tr>
<tr>
<td>predict.CTRLmodel</td>
<td>66</td>
</tr>
<tr>
<td>predict.DBRmodel</td>
<td>67</td>
</tr>
<tr>
<td>predict.EBRmodel</td>
<td>69</td>
</tr>
<tr>
<td>predict.ECCmodel</td>
<td>70</td>
</tr>
<tr>
<td>predict.EPSmodel</td>
<td>71</td>
</tr>
<tr>
<td>predict.ESLmodel</td>
<td>73</td>
</tr>
<tr>
<td>predict.HOMERmodel</td>
<td>74</td>
</tr>
<tr>
<td>predict.LIFTmodel</td>
<td>75</td>
</tr>
<tr>
<td>predict.LPmodel</td>
<td>76</td>
</tr>
<tr>
<td>predict.MBRmodel</td>
<td>77</td>
</tr>
<tr>
<td>predict.MLKNmodel</td>
<td>78</td>
</tr>
<tr>
<td>predict.NSmodel</td>
<td>79</td>
</tr>
<tr>
<td>predict.PPTmodel</td>
<td>80</td>
</tr>
<tr>
<td>predict.PruDentmodel</td>
<td>81</td>
</tr>
<tr>
<td>predict.PSmodel</td>
<td>82</td>
</tr>
<tr>
<td>predict.RAkELmodel</td>
<td>83</td>
</tr>
<tr>
<td>predict.RDBRmodel</td>
<td>84</td>
</tr>
<tr>
<td>predict.RPCmodel</td>
<td>85</td>
</tr>
<tr>
<td>print.BRmodel</td>
<td>87</td>
</tr>
<tr>
<td>print.BRPmodel</td>
<td>87</td>
</tr>
<tr>
<td>print.CCmodel</td>
<td>88</td>
</tr>
<tr>
<td>print.CLRmodel</td>
<td>88</td>
</tr>
<tr>
<td>print.CTRLmodel</td>
<td>89</td>
</tr>
<tr>
<td>print.DBRmodel</td>
<td>89</td>
</tr>
<tr>
<td>print.EBRmodel</td>
<td>90</td>
</tr>
<tr>
<td>print.ECCmodel</td>
<td>90</td>
</tr>
<tr>
<td>print.EPSmodel</td>
<td>91</td>
</tr>
<tr>
<td>print.ESLmodel</td>
<td>91</td>
</tr>
<tr>
<td>print.kFoldPartition</td>
<td>92</td>
</tr>
<tr>
<td>print.LIFTmodel</td>
<td>92</td>
</tr>
<tr>
<td>print.LPmodel</td>
<td>93</td>
</tr>
<tr>
<td>print.majorityModel</td>
<td>93</td>
</tr>
<tr>
<td>print.MBRmodel</td>
<td>94</td>
</tr>
<tr>
<td>print.mlconfmat</td>
<td>94</td>
</tr>
<tr>
<td>print.MLKNmodel</td>
<td>95</td>
</tr>
<tr>
<td>print.mlresult</td>
<td>95</td>
</tr>
<tr>
<td>print.NSmodel</td>
<td>96</td>
</tr>
<tr>
<td>print.PPTmodel</td>
<td>96</td>
</tr>
<tr>
<td>print.PruDentmodel</td>
<td>97</td>
</tr>
<tr>
<td>print.PSmodel</td>
<td>97</td>
</tr>
</tbody>
</table>
Join two multi-label confusion matrix

## 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.mlconfmat

Convert a multi-label Confusion Matrix to matrix

**Description**

Convert a multi-label Confusion Matrix to matrix

**Usage**

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

**Arguments**

- `x` The mlconfmat
- `...` passed to as.matrix
as.matrix.mlresult  
\textit{Convert a mlresult to matrix}

\textbf{Description}

Convert a mlresult to matrix

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'mlresult'
as.matrix(x, ...)
\end{verbatim}

\textbf{Arguments}

- \texttt{x}  
  The mlresult object
- \texttt{...}  
  ignored

\textbf{Value}

matrix

as.mlresult  
\textit{Convert a matrix prediction in a multi label prediction}

\textbf{Description}

Convert a matrix prediction in a multi label prediction

\textbf{Usage}

\begin{verbatim}
as.mlresult(predictions, probability = TRUE, ...)
\end{verbatim}

\textbf{Arguments}

- \texttt{predictions}  
  a Matrix or data.frame contained the scores/probabilities values. The columns are the labels and the rows are the examples.
- \texttt{probability}  
  A logical value. If \texttt{TRUE} the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1. (Default: \texttt{TRUE})
- \texttt{...}  
  ignored
- \texttt{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.
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**

```r
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

---

baseline

*Baseline reference for multilabel classification*

**Description**

Create a baseline model for multilabel classification.

**Usage**

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

**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').
- **...**
  - not used
Details

Baseline is a na"ive 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.

ranking-loss Predict a ranking based on the most frequent labels.

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("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**: The number of cores to parallelize the training. (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

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()`, `dbp()`, `ebr()`, `ecc()`, `eps()`, `esl()`, `homer()`, `lift()`, `lp()`, `mbr()`, `ns()`, `ppt()`, `prudent()`, `ps()`, `rakel()`, `rdb()`, `rpc()`

Examples

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

```r
## Not run:
# Use SVM as base algorithm
model <- br(toyml, "SVM")
pred <- predict(model, toyml)
```
# Change the base algorithm and use 4 CORES
model <- br(toyml[1:50], 'RF', cores = 4, seed = 123)

# Set a parameters for all subproblems
model <- br(toyml, '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.algorithm", "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.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("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()`, `esl()`, `homer()`, `lift()`, `lp()`, `mbr()`, `ns()`, `ppt()`, `prudent()`, `ps()`, `rakel()`, `rdbr()`, `rpc()`

Other Stacking methods: `mbr()`

Examples

```r
# 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)
```

---

**cc**

*Classifier Chains for multi-label Classification*

Description

Create a Classifier Chains model for multilabel classification.

Usage

```r
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


See Also

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

```r
clr(
  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`: 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(), dbre(), ebr(), ecc(), eps(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), prudent(), ps(), rakel(), rdb(), rpc()

Other Pairwise methods: rpc()

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

```r
## Not run:

compute_multilabel_predictions

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

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

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
create_holdout_partition

Create a holdout partition based on the specified algorithm

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.

probability A logical value. If TRUE the predicted values are the score between 0 and 1, otherwise the values are bipartition 0 or 1.

Value

A mlresult with computed predictions.

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 mldr 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 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

Other sampling: create_kfold_partition(), create_random_subset(), create_subset()
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:

Examples

```r
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$measures$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
}
dataset <- create_holdout_partition(toyml, method="sequential_split")
```
create_kfold_partition

**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**

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

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

## 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))
CTRL model for multi-label Classification

Description

Create a binary relevance with ConTRolled Label correlation exploitation (CTRL) model for multi-label 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: `getOption("utiml.base.algorithm", "SVM")`)  
- **m**: The max number of Binary Relevance models used in the binary ensemble. (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: `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)`)
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(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), prudent(), ps(), rakel(), rdbr(), rpc()

Examples

```r
## Not run:
model <- ctrl(toyml, "RANDOM")
pred <- predict(model, toyml)

# 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)
```
Multilabel cross-validation

**Description**

Perform the cross validation procedure for multi-label learning.

**Usage**

```r
cv(
  mdata,
  method,
  ...,  # Additional parameters required by the method.
  cv.folds = 10,
  cv.sampling = c("random", "iterative", "stratified"),
  cv.results = FALSE,
  cv.predictions = FALSE,
  cv.measures = "all",
  cv.cores = getOption("utiml.cores", 1),
  cv.seed = getOption("utiml.seed", NA)
)
```

**Arguments**

- `mdata`: A mldr dataset.
- `method`: The multi-label classification method. It also accepts the name of the method as a string.
- `...`: Additional parameters required by the method.
- `cv.folds`: Number of folds. (Default: 10)
- `cv.sampling`: 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. (Default: "random")
- `cv.results`: Logical value indicating if the folds results should be reported (Default: FALSE).
- `cv.predictions`: Logical value indicating if the predictions should be reported (Default: FALSE).
- `cv.measures`: The measures names to be computed. Call `multilabel_measures()` to see the expected measures. You can also use "bipartition", "ranking", "label-based", "example-based", "macro-based", "micro-based" and "label-problem" to include a set of measures. (Default: "all").
- `cv.cores`: The number of cores to parallelize the cross validation procedure. (Default: `getOption("utiml.cores", 1)`) 
- `cv.seed`: An optional integer used to set the seed. (Default: `getOption("utiml.seed", NA)`)
Value

If cv.results and cv.prediction are FALSE, the return is a vector with the expected multi-label measures, otherwise, a list contained the multi-label and the other expected results (the label measures and/or the prediction object) for each fold.

See Also

Other evaluation: multilabel_confusion_matrix(), multilabel_evaluate(), multilabel_measures()

Examples

# Run 10 folds for BR method
res1 <- cv(toyml, br, base.algorithm="RANDOM", cv.folds=10)

# Run 3 folds for RAKEL method and get the fold results and the prediction
res2 <- cv(mdata=toyml, method="rakel", base.algorithm="RANDOM", k=2, m=10, cv.folds=3, cv.results=TRUE, cv.predictions=TRUE)

---

dbr

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

dbr(
  mdata,
  base.algorithm = getOption("utiml.base.algorithm", "SVM"),
  estimate.models = TRUE,
  ...
)

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"))
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()`, `esl()`, `homer()`, `lift()`, `lp()`, `mbr()`, `ns()`, `ppt()`, `prudent()`, `ps()`, `rakel()`, `rdbr()`, `rpc()`

### Examples  
```
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

ebr(
  mdata,  # A mldr dataset used to train the binary models.
  base.algorithm = getOption("utiml.base.algorithm", "SVM"),
  m = 10,  # The number of Binary Relevance models used in the ensemble. (Default: 10)
  subsample = 0.75,  # 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 = 0.5,  # A value between 0.1 and 1 to determine the percentage of attributes that must be
                   # used for each classifier. (Default: 0.50)
  replacement = TRUE,  # 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 = getOption("utiml.cores", 1),  # The number of cores to parallelize the training. Values higher than 1 require the
  seed = getOption("utiml.seed", NA)  # An optional integer used to set the seed. This is useful when the method is run
                                           # in parallel. (Default: options("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 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)
... 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 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()`, `esi()`, `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)

## 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)

## End(Not run)
```

---

**ecc**

*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("uti ml.base.algorithm", "SVM"),
    m = 10,
    subsample = 0.75,
    attr.space = 0.5,
    replacement = TRUE,
    ...,
    cores = getOption("uti ml.cores", 1),
    seed = getOption("uti ml.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("uti ml.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("uti ml.cores", 1))
seed An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options("uti ml.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 utilml.mc.set.seed to FALSE.

References


See Also

Other Transformation methods: brplus(), br(), cc(), clr(), ctrl(), dbr(), ebr(), eps(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), prudent(), ps(), ral(), rdb(), rpc()

Other Ensemble methods: ebr(), eps()

Examples

# 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
model1 <- ecc(toyml, cores=4, seed=123)

## End(Not run)
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 mldr dataset used to train the binary models.
- `base.algorithm` A string with the name of the base algorithm. (Default: `getOption("utiml.base.algorithm", "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: `getOption("utiml.cores", 1)`)  
- `seed` An optional integer used to set the seed. (Default: `getOption("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(), esl(), 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
model <- eps(toyml, "RF", m=15, subsample=0.4, p=4, strategy="B", b=4)
## End(Not run)
```

---

esl  

**Ensemble of Single Label**

Description

Create an Ensemble of Single Label model for multilabel classification.

Usage

```r
esl(  
mdata,  
base.algorithm = getOption("utiml.base.algorithm", "SVM"),  
m = 10,  
w = 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"))
- `m`: The number of members used in the ensemble. (Default: 10)
- `w`: The weight given to the choice of the less frequent labels. When it is 0, the labels will be random choose, when it is 1 the complement of the label frequency is used as the probability to choose each label. Values greater than 1 will privilege the less frequent labels. (Default: 1)
- `...`: Others arguments passed to the base algorithm for all subproblems
fill_sparse_mldata

fill_sparse_mldata

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

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

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

## S3 method for class 'Var'
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
- mresult: Fixed Threshold for mresult

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('Varlbl1', 'Varlb2', 'Varlb3'))
)

# Use 0.5 as threshold
fixed_threshold(result)

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

foodtruck

Foodtruck multi-label dataset.

Description

The foodtruck multi-label dataset is a real multi-label dataset, which uses habits and personal information to predict food truck cuisines.

Usage

foodtruck

Format

A mldr object with 407 instances, 21 features and 12 labels:
Details

General Information

- Cardinality: 2.28
- Density: 0.19
- Distinct multi-labels: 117
- Number of single labelsets: 74
- Max frequency: 114

Source


homer

Hierarchy Of Multilabel classifier (HOMER)

Description

Create a Hierarchy Of Multilabel classifier (HOMER).

Usage

homer(
  mdata,
  base.algorithm = getOption("utiml.base.algorithm", "SVM"),
  clusters = 3,
  method = c("balanced", "clustering", "random"),
  iteration = 100,
  ...
  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.
... 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
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.

References

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

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

## Not run:
##Change default configurations
model <- homer(toyml, "RF", clusters=5, method="clustering", iteration=10)

## End(Not run)
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**

```r
lcard_threshold(prediction, cardinality, probability = FALSE)
```

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

```r
## 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)
```

---

**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 mldr dataset used to train the binary models.
- `base.algorithm`: A string with the name of the base algorithm. (Default: `getOption("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: `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)`)  

**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.
lp

Label Powerset for multi-label Classification

Description

Create a Label Powerset model for multilabel classification.

Usage

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))

Examples

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)
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()`, `esl()`, `homer()`, `lift()`, `mbr()`, `ns()`, `ppt()`, `prudent()`, `ps()`, `rakel()`, `rdb()`, `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("utiml.base.algorithm", "SVM"),
  folds = 1,
  phi = 0,
  ...
  predict.params = list(),
  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"))
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("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 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(), db(), ebr(), ecc(), eps(), esl(), homer(), lift(), lp(), ns(), ppt(), prudent(), ps(), rakel(), rdbr(), rpc().
Other Stacking methods: brplu()
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)
```

**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)

#### 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
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
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
mldata(mdata)
Arguments

mdata
A mldr dataset.

Value

A mldr object

Examples

toyml <- mldata(toyml)

Description

Create a ML-KNN classifier to predict multi-label data. It is a multi-label lazy learning, which is derived from the traditional K-nearest neighbor (KNN) algorithm. For each unseen instance, its K nearest neighbors in the training set are identified and based on statistical information gained from the label sets of these neighboring instances, the maximum a posteriori (MAP) principle is utilized to determine the label set for the unseen instance.

Usage

mlknn(
mdata,
k = 10,
s = 1,
distance = "euclidean",
...,
cores = getOption("utiml.cores", 1),
seed = getOption("utiml.seed", NA)
)

Arguments

mdata          A mldr dataset used to train the binary models.
k             The number of neighbors. (Default: 10)
s             Smoothing parameter controlling the strength of uniform prior. When it is set to be 1, we have the Laplace smoothing. (Default: 1).
distance     The name of method used to compute the distance. See dist to the list of options. (Default: "euclidian")
...          Not used.
cores        Ignored because this method does not support multi-core.
seed          Ignored because this method is deterministic.
Value

An object of class \texttt{MLKNNmodel} containing the set of fitted models, including:

- **labels** A vector with the label names.
- **prior** The prior probability of each label to occur.
- **posterior** The posterior probability of each label to occur given that k neighbors have it.

References


Examples

```r
model <- mlknn(toyml, k=3)
pred <- predict(model, toyml)
```

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 \texttt{mltrain} documentation.

Usage

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

Arguments

- **model** An object model returned by some \texttt{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.
How to create a new prediction base method

First it 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

```r
# 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')
}
## End(Not run)
```

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, ...)

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.

How to create a new train base method

First, 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 mlconmat 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.
multilabel_evaluate

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.

Zi The total of positive predictions for each label.

Yi The total of positive expected for each label.

TPl The total of True Positive predictions for each label.

FPi The total of False Positive predictions for each label.

TNl The total of True Negative predictions for each label.

FNI The total False Negative predictions for each label.

See Also

Other evaluation: cv(), multilabel_evaluate(), multilabel_measures()

Examples

## 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)

# Number of wrong predictions for each label
errors <- mlconfmat$FPi + mlconfmat$FNI

# Examples predict with all labels
which(mlconfmat$Zi == toyml$measures$num.labels)

# You can join one or more mlconfmat
part1 <- create_subset(toyml, 1:50)
part2 <- create_subset(toyml, 51:100)
confmat1 <- multilabel_confusion_matrix(part1, prediction[1:50, ])
confmat2 <- multilabel_confusion_matrix(part2, prediction[51:100, ])
mlconfmat <- confmat1 + confmat2

## End(Not run)

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

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

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

```r  
## 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 can also use "bipartition", "ranking", "label-based", "example-based", "macro-based", "micro-based" and "label-problem" to include a set of measures. (Default: "all").
- `labels`: Logical value defining if the label results should be also returned. (Default: FALSE)

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


See Also

Other evaluation: `cv()`, `multilabel_confusion_matrix()`, `multilabel_measures()`
multilabel_measures

Examples

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

# Compute all measures
multilabel_evaluate(toyml, prediction)
multilabel_evaluate(toyml, prediction, labels=TRUE) # Return a list

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

# Compute multiples 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)
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

```r
multilabel_measures()
```

Value

array of character contained the measures names.

See Also

Other evaluation: cv(), multilabel_confusion_matrix(), multilabel_evaluate()

Examples

```r
multilabel_measures()
```
multilabel_prediction  

Create a mlresult object

Description

Create a mlresult object

Usage

```r
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

```r
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**
```r
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**
```r
norm.toy <- normalize_mldata(toyml)
```

---

**ns**

**Nested Stacking for multi-label Classification**

**Description**
Create a Nested Stacking model for multilabel classification.

**Usage**
```r
s(  
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 NSmodel 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

classification. In Workshop of Lernen, Wissen & Adaptivitat (LWA 2013) (pp. 162-169). Bamberg,
Germany.

See Also

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

Examples

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
```r
model <- ns(toyml, 'KNN', k=5)
## End(Not run)
```

---

**partition_fold**  
Create the multi-label dataset from folds

### Description

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

### Usage

```r
partition_fold(kfold, n, has.validation = FALSE)
```

### Arguments

- `kfold`: A `kFoldPartition` object obtained from use of the method `create_kfold_partition`.
- `n`: The number of the fold to separated train and test subsets.
- `has.validation`: Logical value that indicate if a validation dataset will be used. (Default: `FALSE`)

### Value

A list contained train and test mldr dataset:
- `train`: The mldr dataset with train examples, that includes all examples except those that are in test and validation samples.
- `test`: The mldr dataset with test examples, defined by the number of the fold.
- `validation`: Optionally, only if `has.validation = 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**

```r
pcut_threshold(prediction, ratio, probability = FALSE)
```

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

```r
## 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

```
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.
predict.BASELINEmodel

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()`, `dbr()`, `ebr()`, `ecc()`, `eps()`, `esl()`, `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.BASELINEmodel

**Predict Method for BASELINE**

#### Description

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

#### Usage

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

#### Arguments

- **object**: Object of class `BASELINEmodel`.
- **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))
- **...**: not used.
predict.BRmodel

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

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)

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("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

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

# 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)

# Passing a specif 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

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

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>Object of class 'CCmodel'.</th>
</tr>
</thead>
<tbody>
<tr>
<td>newdata</td>
<td>An object containing the new input data. This must be a matrix, data.frame or a mldr object.</td>
</tr>
<tr>
<td>probability</td>
<td>Logical indicating whether class probabilities should be returned. (Default: getOption(&quot;utiml.use.probs&quot;,TRUE))</td>
</tr>
<tr>
<td>...</td>
<td>Others arguments passed to the base algorithm prediction for all subproblems.</td>
</tr>
<tr>
<td>cores</td>
<td>Ignored because this method does not support multi-core.</td>
</tr>
<tr>
<td>seed</td>
<td>An optional integer used to set the seed. (Default: options(&quot;utiml.seed&quot;,NA))</td>
</tr>
</tbody>
</table>
Value

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

Note

The Classifier Chains prediction can not 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 specif parameter for SVM predict algorithm
pred <- predict(model, toyml, na.action = na.fail)

## End(Not run)
```
predict.CTRLmodel

Arguments

object Object of class 'CLRmodel'.
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

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

## 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)
)
```
predict.DBRmodel

**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: 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**

CTRL

**Examples**

```r
## Not run:
model <- ctrl(toyml, "RANDOM")
pred <- predict(model, toyml)

# 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 mldr 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 `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.

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)
)
```

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

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

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

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

```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 mlresult, 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")
```

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.ESLmodel

**Predict Method for Ensemble of Single Label**

**Description**
This function predicts values based upon a model trained by `esl`.

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

**Arguments**
- `object`: Object of class 'ESLmodel'.
- `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**
- Ensemble of Single Label (ESL)

**Examples**
```r
model <- esl(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)`)  
- `...`: 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**

*Hierarchy Of Multilabel classifiER (HOMER)*

**Examples**

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

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 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

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: `getOption("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

**Description**

This function predicts values based upon a model trained by \texttt{mbr}.

**Usage**

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

**Arguments**

- **object**: Object of class `\texttt{MBRmodel}`.
- **newdata**: An object containing the new input data. This must be a matrix, data.frame or a \texttt{mldr} object.
- **probability**: Logical indicating whether class probabilities should be returned. (Default: \texttt{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: \texttt{options("utiml.cores",1)})
- **seed**: An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: \texttt{options("utiml.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)

## End(Not run)

---

**predict.MLKNNmodel**  
*Predict Method for ML-KNN*

**Description**

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

**Usage**

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

**Arguments**

- **object**: Object of class 'MLKNNmodel'.
- **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**: Ignored because this method does not support multi-core.

**Value**

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

**See Also**

ML-KNN
predict.NSmodel

Examples

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

---

**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)

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

## End(Not run)
```

### predict.PPTmodel

**Predict Method for Pruned Problem Transformation**

#### Description

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

#### Usage

```r
## 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: `getOption("utiml.seed", NA)`)  

#### Value

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

See Also

Pruned Problem Transformation (PPT)

Examples

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

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.PSmodel

### Predict Method for Pruned Set Transformation

**Description**

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

**Usage**

```r
## 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)`)
- `...`: 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`. 
predict.RAkELmodel

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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Object of class 'RAkELmodel'.</td>
</tr>
<tr>
<td>newdata</td>
<td>An object containing the new input data. This must be a matrix, data.frame or a mldr object.</td>
</tr>
<tr>
<td>probability</td>
<td>Logical indicating whether class probabilities should be returned. (Default: getOption(&quot;utiml.use.probs&quot;,TRUE))</td>
</tr>
<tr>
<td>cores</td>
<td>The number of cores to parallelize the prediction. Values higher than 1 require the parallel package. (Default: options(&quot;utiml.cores&quot;,1))</td>
</tr>
<tr>
<td>seed</td>
<td>An optional integer used to set the seed. This is useful when the method is run in parallel. (Default: options(&quot;utiml.seed&quot;,NA))</td>
</tr>
</tbody>
</table>

Value

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

See Also

Random k Labelsets (RAkEL)
predict.RDBRmodel

Examples

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

## 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 mlr object.
estimative  A matrix containing the bipartition result of other multi-label classification algorithm or an mlresult 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
data <- rdbr(toyml)
pred <- predict(data, toyml)

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

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

# Using other classifier (EBR) to make the labels estimatives
estimative <- predict(ebr(toyml), toyml, probability = FALSE)
data <- rdbr(toyml, estimate.models = FALSE)
pred <- predict(data, 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 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 <- rpc(toyml, "RANDOM")
pred <- predict(model, toyml)
```

```r
## Not run:
```
print.BRmodel

**Description**
Print BR model

**Usage**
```r
## S3 method for class 'BRmodel'
print(x, ...)
```

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

---

print.BRPmodel

**Description**
Print BRP model

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

**Arguments**
- `x` The brp model
- `...` ignored
print.CCmodel

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

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

Arguments

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

Description

Print CTRL model

Usage

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

Arguments

x The ctrlmodel
...

print.DBRmodel

Description

Print DBR model

Usage

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

Arguments

x The dbr model
...

ignored
print.EBRmodel | Print EBR model

Description
Print EBR model

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

Arguments
x The ebr model
... ignored

print.ECCmodel | Print ECC model

Description
Print ECC model

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

Arguments
x The ecc model
... ignored
print.EPSmodel

Description

Print EPS model

Usage

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

Arguments

x The ps model
...

print.ESLmodel

Description

Print ESL model

Usage

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

Arguments

x The esl model
...

ignored
print.kFoldPartition  

Print a kFoldPartition object

Description

Print a kFoldPartition object

Usage

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

Arguments

x  The kFoldPartition object
...

print.LIFTmodel  

Print LIFT model

Description

Print LIFT model

Usage

## 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*

---

**Description**

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*

---

**Description**

Print a Multi-label Confusion Matrix

**Usage**

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

**Arguments**

- `x`  
  The mlconfmat

- `...`  
  ignored
**Description**

Print MLKNN model

**Usage**

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

**Arguments**

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

---

**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  

**Print NS model**

**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**

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

**Arguments**

- `x`  
The ppt model
- `...`  
  ignored
**print.PruDentmodel**  
*Print PruDent model*

**Description**

Print PruDent model

**Usage**

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

**Arguments**

- `x` The prudent model
- `...` ignored

---

**print.PSmodel**  
*Print PS model*

**Description**

Print PS model

**Usage**

```r
## 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
...

print.randomModel

Print Random model

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**

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

**Arguments**

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

--

print.RPCmodel  

**Description**

Print RPC model

**Usage**

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

**Arguments**

- `x`: The br model
- `...`: ignored
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: `getOption("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: `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)`)  

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(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), ps(), rakel(), rdb(), 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)
## End(Not run)
```

### Description

Create a Pruned Set model for multilabel classification.

### Usage

```r
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 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"))
ps

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


See Also

Other Transformation methods: brplus(), br(), cc(), cl(), ctrl(), db(), ebr(), ecc(), eps(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), prudent(), rakel(), rdbr(), rpc()
Other Powerset: eps(), lp(), ppt(), rakel()

Examples

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

## Not run:
## Change default configurations
model <- ps(toyml, "RF", p=4, strategy="B", b=4)

## End(Not run)
Description

Create a RAkEL model for multilabel classification.

Usage

```r
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: `getOption("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: `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)`)

Details

RAAndom 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.

References


See Also

Other Transformation methods: brplus(), br(), cc(), clr(), ctrl(), dbr(), ebr(), ecc(),
eps(), esl(), homer(), lift(), lp(), mbr(), ns(), ppt(), prudent(), ps(), rdb(), 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

```r
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 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"))

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))

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: \texttt{brplus()}, \texttt{br()}, \texttt{cc()}, \texttt{clr()}, \texttt{ctrl()}, \texttt{dbr()}, \texttt{ebr()}, \texttt{ecc()}, \texttt{eps()}, \texttt{esl()}, \texttt{homer()}, \texttt{lift()}, \texttt{lp()}, \texttt{mbr()}, \texttt{ns()}, \texttt{ppt()}, \texttt{prudent()}, \texttt{ps()}, \texttt{rakel()}, \texttt{rpc()}

Examples

```r
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: \texttt{fill\_sparse\_mldata()}, \texttt{normalize\_mldata()}, \texttt{remove\_labels()}, \texttt{remove\_skewness\_labels()}, \texttt{remove\_unique\_attributes()}, \texttt{remove\_unlabeled\_instances()}, \texttt{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

```
remove_labels(mdata, 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)
```
remove_unique_attributes

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()

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 mlr dataset to remove the instances.

Value
a new mlr 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)

classification = predict(mdata)

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 mlr dataset to remove.
ordinal.attributes Not yet, but it will be used to specify which attributes need to be replaced.
Value

a new mlr 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

new.toy <- toyml
new.column <- as.factor(sample(c("a","b","c"), 100, replace = TRUE))
new.toy$dataset$att10 <- new.column
head(replace_nominal_attributes(new.toy))
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()`, `esl()`, `homer()`, `lift()`, `lp()`, `mbr()`, `ns()`, `ppt()`, `prudent()`, `ps()`, `rakel()`, `rdbr()`

Other Pairwise methods: `clr()`

Examples

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

```r
## 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)
)
```

```r
## Default S3 method:
scut_threshold(
  prediction,
  expected,
  loss.function = NA,
```
```r
scut_threshold

  cores = getOption("utiml.cores", 1)

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

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

  partment of Computer Science, National Taiwan University.
  Threshold Selection Methods. In First International Workshop on Learning over Multiple Contexts
  (LMCE) at ECML-PKDD 2014.
```
See Also

Other threshold: fixed_threshold(), lcard_threshold(), mcut_threshold(), pcut_threshold(), rcut_threshold(), subset_correction()

Examples

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:
# 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

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)

summary.mltransformation

Summary method for mltransformation

Description

Summary method for mltransformation

Usage

## S3 method for class 'mltransformation'
summary(object, ...)

Arguments

object A transformed dataset
... additional arguments affecting the summary produced.
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

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  

**utiml: Utilities for Multi-Label Learning**

---

**Description**

The utiml package is a framework for the application of classification algorithms to multi-label data. Like the well known MULAN used with Weka, it provides a set of multi-label procedures such as sampling methods, transformation strategies, threshold functions, pre-processing techniques and evaluation metrics. The package was designed to allow users to easily perform complete multi-label classification experiments in the R environment.

**Details**

Currently, the main methods supported are:

1. **Classification methods**: ML Baselines, Binary Relevance (BR), BR+, Classifier Chains, Calibrated Label Ranking (CLR), ConTRolled Label correlation exploitation (CTRL), Dependent Binary Relevance (DBR), Ensemble of Binary Relevance (EBR), Ensemble of Classifier Chains (ECC), Ensemble of Pruned Set (EPS), Hierarchy Of Multilabel Classifier (HOMER), Label specific Features (LIFT), Label Powerset (LP), Meta-Binary Relevance (MBR or 2BR), Multi-label KNN (ML-KNN), Nested Stacking (NS), Pruned Problem Transformation (PPT), Pruned and Confident Stacking Approach (Prudent), Pruned Set (PS), Random k-labelsets (RAkEL), Recursive Dependent Binary Relevance (RDBR), Ranking by Pairwise Comparison (RPC)

2. **Evaluation methods**: Performing a cross-validation procedure, 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, Cardinality 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

---

**`.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**

```r
## 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
  foodtruck, 35
  toyml, 116
  +.mlconfmat, 4
  [.mlresult, 118
  as.bipartition, 5, 118
  as.matrix.mlconfmat, 5
  as.matrix.mlresult, 6
  as.mlresult, 6, 118
  as.probability, 7
  as.ranking, 8, 118

Baseline, 61
baseline, 8, 60
Binary Relevance (BR), 62, 66, 86, 117
br, 9, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 61, 101, 102, 104,
  107, 112
BR+, 63, 117
brplus, 10, 11, 13, 15, 23, 26, 28, 30, 32, 33,
  37, 41–43, 56, 60, 101, 102, 104,
  107, 112

Calibrated Label Ranking (CLR), 117
Cardinality threshold, 118
cc, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 101, 102, 104, 107,
  112
Classifier Chains, 117
Classifier Chains (CC), 65
clr, 10, 12, 13, 14, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 65, 101, 102, 104,
  107, 112
compute_multilabel_predictions, 15
Confusion Matrix, 117
ConTrolled Label correlation
  exploitation (CTRL), 117
Create holdout partitions, 117
Create random subset, 117

Create subset, 117
create_holdout_partition, 16, 19–21
create_kfold_partition, 17, 18, 20, 21, 157
create_random_subset, 17, 19, 20, 21
create_subset, 17, 19, 20, 21
CTRL, 67
dCtrl, 10, 12, 13, 15, 22, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 66, 101, 102, 104,
  107, 112
cv, 24, 51–53
dbr, 10, 12, 13, 15, 23, 25, 28, 30, 32, 33, 37,
  41–43, 56, 60, 101, 102, 104, 107,
  112
Dependent Binary Relevance (DBR), 68,
  107, 117
dist, 46
ebr, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 69, 101, 102, 104,
  107, 112
ecc, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 70, 101, 102, 104, 107,
  112
Ensemble of Binary Relevance (EBR), 70,
  117
Ensemble of Classifier Chains (ECC),
  71, 117
Ensemble of Pruned Set (EPS), 72, 117
Ensemble of Single Label (ESL), 73
esps, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37,
  41–43, 56, 60, 71, 101, 102, 104,
  107, 112
esl, 10, 12, 13, 15, 23, 26, 28, 30, 32, 32, 37,
  41–43, 56, 60, 73, 101, 102, 104,
  107, 112
Evaluate, 117
Fill sparse data, 117
fill_sparse_mldata, 33, 55, 107–111
Fixed threshold, 118
fixed_threshold, 34, 39, 45, 59, 105, 114, 115
foodtruck, 35

Hierarchy Of Multilabel classifier (HOMER), 74, 117
homer, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 36, 41–43, 56, 60, 74, 101, 102, 104, 107, 112
How to create the datasets from folds, 19
is.bipartition, 38
is.probability, 38

Label Power set (LP), 76, 117
Label specific Features (LIFT), 117
lcard_threshold, 35, 39, 45, 59, 105, 114, 115
LIFT, 75
lift, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37, 40, 42, 43, 56, 60, 75, 101, 102, 104, 107, 112
lp, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37, 41, 43, 56, 60, 76, 101, 102, 104, 107, 112
mbr, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37, 41, 42, 43, 56, 60, 101, 102, 104, 107, 112
MCUT, 118
mcut_threshold, 35, 39, 44, 59, 105, 114, 115
merge_mlconfmat, 45
ML Baselines, 117
mldata, 45
mldr, 118
mlknn, 46
mlpredict, 47, 48
mltrain, 47, 48
multilabel_confusion_matrix, 25, 50, 52, 53
multilabel_evaluate, 25, 51, 51, 53
multilabel_measures, 25, 51, 52, 53
multilabel_prediction, 54, 118

Nested Stacking (NS), 79, 117
Normalize data, 117
normalize_mldata, 34, 55, 107–111

ns, 10, 12, 15, 23, 26, 28, 30, 32, 33, 37, 41–43, 55, 60, 101, 102, 104, 107, 112
Partition fold, 117
partition_fold, 57
PCUT, 118
pcut_threshold, 35, 39, 45, 58, 105, 114, 115
ppt, 10, 12, 13, 15, 23, 26, 28, 30, 32, 33, 37, 41–43, 56, 59, 80, 101, 102, 104, 107, 112
predict.BASELINEmodel, 60
predict.BRmodel, 61
predict.BRPmodel, 62, 67
predict.CCmodel, 64
predict.CLRmodel, 65
predict.CTRLmodel, 66
predict.DBRmodel, 67, 84
predict.EBRmodel, 69
predict.ECCmodel, 70
predict.EPSmodel, 71
predict.ESLmodel, 73
predict.HOMERmodel, 74
predict.LIFTmodel, 75
predict.LPmodel, 76
predict.MBRmodels, 77
predict.MLKNmodels, 78
predict.NSmodesls, 79
predict.PPTmodel, 80
predict.PruDentmodels, 81
predict.PSmodels, 82
predict.RAkELmodels, 83
predict.RDBRmodel, 84
predict.RPCmodel, 85
print.BRmodel, 87
print.BRPmodel, 87
print.CCmodel, 88
print.CLRmodel, 88
print.CTRLmodel, 89
print.DBRmodel, 89
print.EBRmodel, 90
print.ECCmodel, 90
print.EPSmodel, 91
print.ESLmodel, 91
print.kFoldPartition, 92
print.LIFTmodel, 92
print.LPmodel, 93
print.majorityModel, 93
print.MBRmodel, 94