International Master in Computer Vision

Fundamentals of machine learning for computer vision

Contents

Ensembles

- **Algorithms that use several instances of the same base classifier**. Example: combination of various classification trees.
- The algorithm name (meta-classifier) is provided by the method use to combine the classifiers.
- Example: group of classifiers of the same type trained of a different way, with a voting to decide the output.
- The base classifiers are usually **weak**: do not work very well, but they are simple and their training is fast.
- The combination of weak classifiers is expected to increase the classification quality, developing a **strong** classifier.

Combination of classifiers

- The base classifiers should be **diverse:** each base classifier should learn a different view of the problem, in order to their combination will be strong.
- Diversity among base classifiers is caused by:
  - Training initialization: for example, MLPs with different random weight initialization.
  - Hyper-parameter tuning: combination of MLPs with different number of hidden layers.
  - Training set: different training sets for the base classifiers.
- The combination algorithms can be applied on different types of base classifiers: e.g. bagging of decision trees or KNN.

Types of ensembles

- **Boosting**: the base classifiers are trained on the same training set but with different pattern weightings.
  - **AdaBoost** is the most popular boosting algorithm
- **Bagging**: the base classifiers are trained on different bootstrap samples of the training set.
  - **Random Forest (RF)** is the most popular bagging algorithm that uses random tree base classifiers.

Boosting

- ***Boosts*** the quality of base classifiers, which also use different pattern weights. **Adaboost** (*adaptive boosting*) is the most popular boosting ensemble.
- The training patterns are weighted in a different way for each base classifier:
- Base classifier 1: all patterns have equal weights
- Base classifier 2 to B: each pattern weight is based on the errors of previous base classifiers on that pattern.

Boosting

- Patterns in which the previous base classifier failed increase their weight in order to be well classified by the following base classifier.
- Besides, each base classifier is weighted according to its reliability.
- Output of ensemble $z(\mathbf{x})$ is the weighted sum of the outputs of the base classifiers in the ensemble.
- Combination of B classifiers: $\{z_b(\mathbf{x},\theta_b)\}_{b=1}^{B}$

Adaboost (I)

- Binary classification y,$z(\mathbf{x}) \in \{\pm 1\}$: $z_b(\mathbf{x}, \theta_b) \in \{\pm 1\}$: output of the *b*-th classifier C_b; θ_b: trainable parameters of C_b

$$z(\boldsymbol{x}) = \text{sign}\left[\sum_{b=1}^{B} a_b z_b(\boldsymbol{x}, \boldsymbol{\theta}_b)\right]$$

Output of the first *b* base classifiers

- Cost function to be minimized: $J(y, z(\mathbf{x})) = e^{-yz(\mathbf{x})}$: when $y=z(\mathbf{x})$, $J=e^{-1}$, when $y \neq z(\mathbf{x})$, $J=e$
- The weight w_i^b of $\mathbf{x}_i$ in iteration *b* is $w_i^b = e^{-y_i u_{b-1}(\boldsymbol{x}_i)}$ for $b>1$ and $w_i^1=1$, where u_b is:

$$u_b(\boldsymbol{x}_i) = \sum_{k=1}^{b} a_k z_k(\boldsymbol{x}_i, \boldsymbol{\theta}_k)$$

- Note that $u_b(\mathbf{x}_i) = u_{b-1}(\mathbf{x}_i) + a_b z_b(\mathbf{x}_i, \theta_b)$

a_k is the weight of base classifier C_k

Adaboost (II)

- In the *b*-th *iteration* (corresponding to base classifier C$_b$), parameters a$_b$ and θ$_b$ are calculated as:

$$(a_b, \boldsymbol{\theta}_b) = \underset{a,\boldsymbol{\theta}}{argmin} \left\{ \sum_{i=1}^{N} w_i^b \, \mathrm{e}^{-y_i u_b(\boldsymbol{x}_i)} \right\}$$

Select the a and **θ** that minimize the error.

- Replacing u$_b$(**x**$_i$):

$$(a_b, \boldsymbol{\theta}_b) = \underset{a,\boldsymbol{\theta}}{argmin} \left\{ \sum_{i=1}^{N} w_i^b \exp\left\{ -y_i \left[u_{b-1}(\boldsymbol{x}_i) + a\, z_b(\boldsymbol{x}_i, \boldsymbol{\theta}) \right] \right\} \right\}$$

- Keeping *a* constant, θ$_b$ is calculated during the training of C$_b$:

$$\boldsymbol{\theta}_b = \underset{\boldsymbol{\theta}}{argmin} \left\{ \sum_{i=1}^{N} w_i^b \exp\left[-y_i\, a\, z_b(\boldsymbol{x}_i, \boldsymbol{\theta}) \right] \right\}$$

Adaboost (III)

- The previous expression can be reduced to:

$$\boldsymbol{\theta}_b = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ P_b(\boldsymbol{\theta}) \right\} \qquad P_b(\boldsymbol{\theta}) = \sum_{y_i \neq z_b(\boldsymbol{x}_i, \boldsymbol{\theta})}^{N} w_i^b$$

where $P_b(\theta)$ is the sum of weights of $\mathbf{x}_i$ patterns with $y_i \neq z_b(\mathbf{x}_i, \theta_b)$, classification errors of ensemble $\{C_b\}_{k=1}^{b-1}$

- Once θ_b and $P_b^m = P_b(\theta_b)$ are calculated, a_b is given by:

$$a_b = \underset{a}{\operatorname{argmin}} \left\{ \underline{e^{-a}\left(1 - P_b^m\right) + e^{a} P_b^m} \right\}$$

$$\uparrow$$

$\boxed{\text{Evaluates error, increasing with } P_b^m}$

Adaboost (IV)

- Deriving the expression $e^{-a_b}\left(1-P_b^m\right)+e^{a_b} P_b^m$ and equaling to 0:

$$a_b=\frac{1}{2}\ln\frac{1-P_b^m}{P_b^m}$$

a_b is decreasing with P_b^m

Lower weight for C_b with higher P_b^m

- As we know θ_b and a_b, the weights w_i^{b+1} are calculated by:

$$w_i^{b+1}=\frac{w_i^b \exp\left[-y_i a_b z_b\left(\boldsymbol{x}_i, \boldsymbol{\theta}_b\right)\right]}{Z_b}$$

where Z_b is the normalization factor:

$$Z_b=\sum_{i=1}^{N} w_i^b \exp\left[-y_i a_b z_b\left(\boldsymbol{x}_i, \boldsymbol{\theta}_b\right)\right]$$

- The process goes on to the following base classifier *b*+1 until *b*=*B*.

Adaboost (V)

- The whole adaboost training algorithm with *B* classifiers is:

```
$w_i^1=1$, $i=1..N$
for $b=1:B-1$
    $P_b(\boldsymbol{\theta})=\sum_{y_i \neq z_b(\boldsymbol{x}_i,\boldsymbol{\theta})}^{N} w_i^b$ ; $\theta_b=\text{argmin}_\theta \{P_b(\theta)\}$
    $a_b=\frac{1}{2}\log\left(\frac{1-P_b^m}{P_b^m}\right)$ ; $P_b^m=P_b(\theta_b)$; $Z_b=0$
    for $i=1:N$
        $w_i^{b+1}=w_i^b \exp[-y_i a_b z_b(\mathbf{x}_i,\theta_b)]$; $Z_b=Z_b+w_i^{b+1}$
    endfor
    for $i=1:N$
        $w_i^{b+1}=w_i^{b+1}/Z_b$
    endfor
endfor
```

Training of the *b*-th base classifier

Output *z* of adaboost for test pattern **x** is: $z(\boldsymbol{x})=\text{sign}\left[\sum_{b=1}^{B} a_b z_b(\boldsymbol{x},\boldsymbol{\theta}_b)\right]$

Bagging (I)

- ***Bootstrap aggregating***: several classifiers are trained on different training sets of the same size.
- The patterns of each training set are randomly selected using the **bootstrap** method: selects some training patterns several times (repeated) and other patterns are not selected. Same size as the original training set.
- The base classifiers are diverse due to different training sets.
- **Output**: voting among the base classifiers.

Bagging (II)

- The base classifiers are normally decision trees.
- Bootstrap increases the quality of base classifiers by reducing variance (less fitting to training data) without increasing bias on test patterns.
- The decision trees tend to over fit the training set, but the bagging algorithm introduces diversity.
- So, the ensemble is not so sensible to noisy data and compensates the over fitting of the single decision trees.

Bagging (III)

- Hyper-parameter *B* (*bag size or number of trees*). The classifier quality is not very sensitive to *B* when a value high enough is provided. Tuning is often not required.
- **E.g.: bagging** function in **ipred** package of R: *B*=25 by default. Normally *B*~100-200 depending of the data size.
- We can also determine *B* using a grid-search (using validation set) or using the *out-of-bag error* (OOB): mean error over the training patterns excluded from the boostrap sample. The OOB stabilizes for enough trees.

Bagging (IV)

- **Output of the bagging ensemble**: if $z_b(\mathbf{x})$ is the output of the *b*-th base classifier (*C*>1 classes):

$$z(\boldsymbol{x}) = \underset{l=1\ldots C}{\arg\max} \left\{ \sum_{b=1}^{B} I[z_b(\boldsymbol{x}), l] \right\}$$

$$I(x, y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$$

Voting among the *B* base classifiers

Random forest (I)

- Combination of decision trees to correct over fitting.
- It uses bagging and random selection of features (inputs), leaving some patterns out of the training set.
- In a decision tree, each node divides the feature that most reduces the entropy.
- In bagging, important features are selected by almost all trees, that are not diverse.
- Random Forest adds diversity by using different feature sets, randomly selected.

Random forest (II)

- RF increases diversity using a group of $q<n$ randomly selected features, different in each base classifier (tree).
- Each tree node splits the best feature in its group.
- Less features are used: faster training.
- The number of inputs selected is usually $q=\sqrt{n}$
- The outputs of the B base classifiers are random variables $\{z_b\}_{b=1}^{B}$, with variance σ and correlation ρ: it can be proven that variance of RF is:

$$var\left(\frac{1}{B}\sum_{b=1}^{B} z_b\right)=\left(\frac{1-\rho}{B}+\rho\right)\sigma^2$$

Random forest (III)

- The random selection of features of RF:

1) Increases the bias, but slightly (-)
2) Increases the variance (σ^2) of each tree (-)
3) Reduces the correlation (ρ) among the trees (+) and raises diversity

- The reduction in correlation ρ (see previous page) is the most important of the three terms: it reduces variance and increases the performance of RF compared to individual trees.

Random forest (IV)

```
RF=$\emptyset$; $K$=maximum number of nodes in the tree
**for** $b=1:B$
    S=bootstrap sample of $\{\mathbf{x}_i, y_i\}_{i=1}^N$; T=$\emptyset$; $r=0$
    **repeat**
        $F=\{i_1..i_q\} \subset \{1..n\}$ with $q<n$, random selection
        Select feature $j \in F$ and threshold $t \in V_j$, $V_j=\{x_{ij}\}_{i=1}^N$ so:
            $$(j,t) = \underset{i=1..n,\, k \in V_i}{\arg\max} \{\Delta E_{ik}\}$$
        $r=r+1$
        Create node $n_r$ ($x_j<t$ and $x_j \geq t$)
        T=T$\cup$$\{n_r\}$
    **until** $r>K$
    RF=RF$\cup$T
**endfor**
Output: $z(\mathbf{x}) = \underset{l=1 \ldots C}{\arg\max} \left\{ \sum_{b=1}^{B} I[z_b(\mathbf{x}), l] \right\}$   $I(x,y) = \begin{cases} 1 & x=y \\ 0 & x \neq y \end{cases}$
```

Current tree

No. nodes

Unique values

ΔE_{ik}=entropy gain of feature i with threshold t

Random forest (V)

- Random Forest provides a measure of the importance of each feature.
- Low number of hyper-parameters and low sensitivity to their values:

1) Number of decision trees B.

2) Number of features q to use in each node.

3) Minimum number of training patterns to split a node.

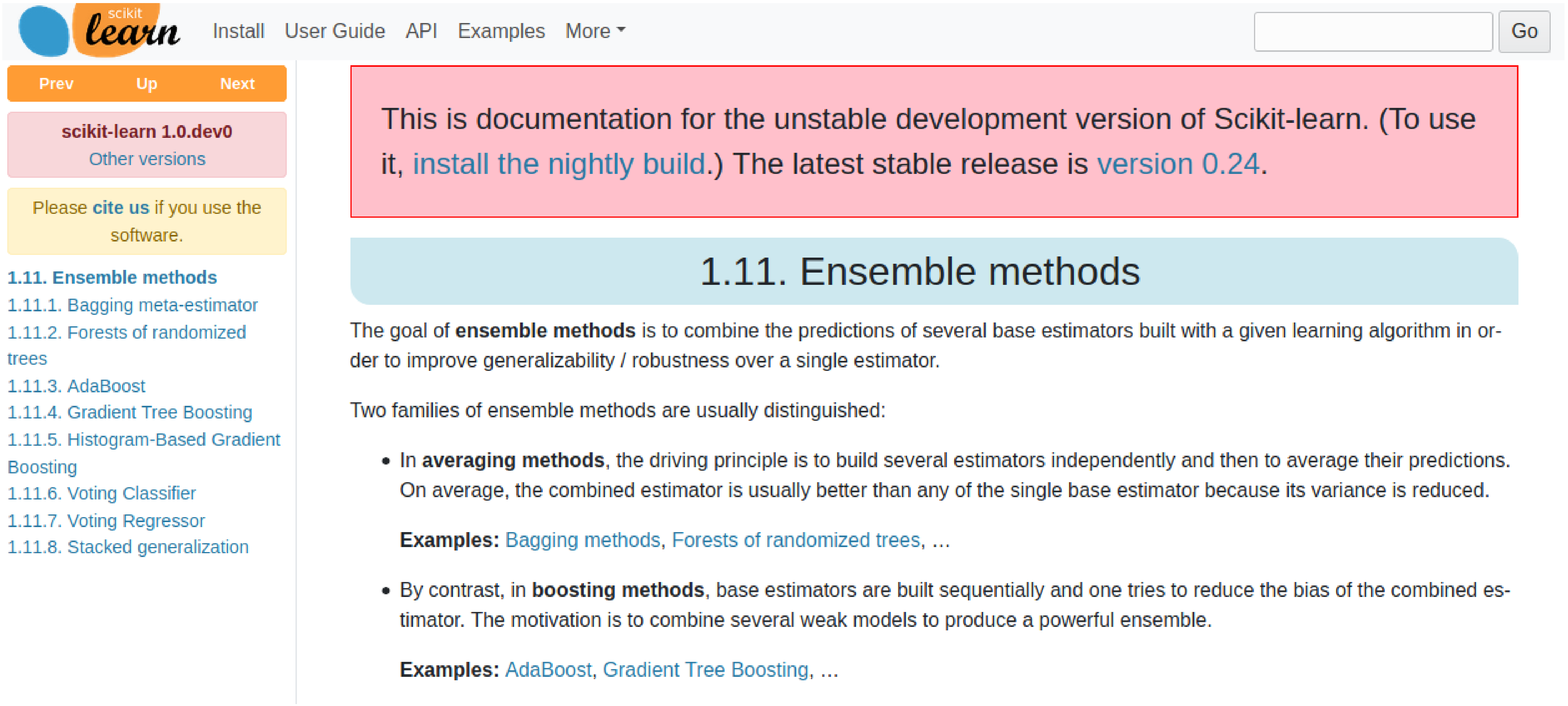
Random forest (VI)

- Increasing *B* does not increase over fitting.
- Very parallelizable.
- It requires low data pre-processing.
- The use of $q<n$ features for RF and $N'<N$ patterns for bootstrapping is efficient with big data.
- Normally very good results: *state-of-the-art* classifier

Python

- **Scikit-learn** package in Python: **ensembles** module

https://scikit-learn.org/stable/modules/ensemble.html



scikit-learn 1.0.dev0
Other versions

Please **cite us** if you use the software.

This is documentation for the unstable development version of Scikit-learn. (To use it, install the nightly build.) The latest stable release is version 0.24.

1.11. Ensemble methods

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

  Examples: Bagging methods, Forests of randomized trees, …

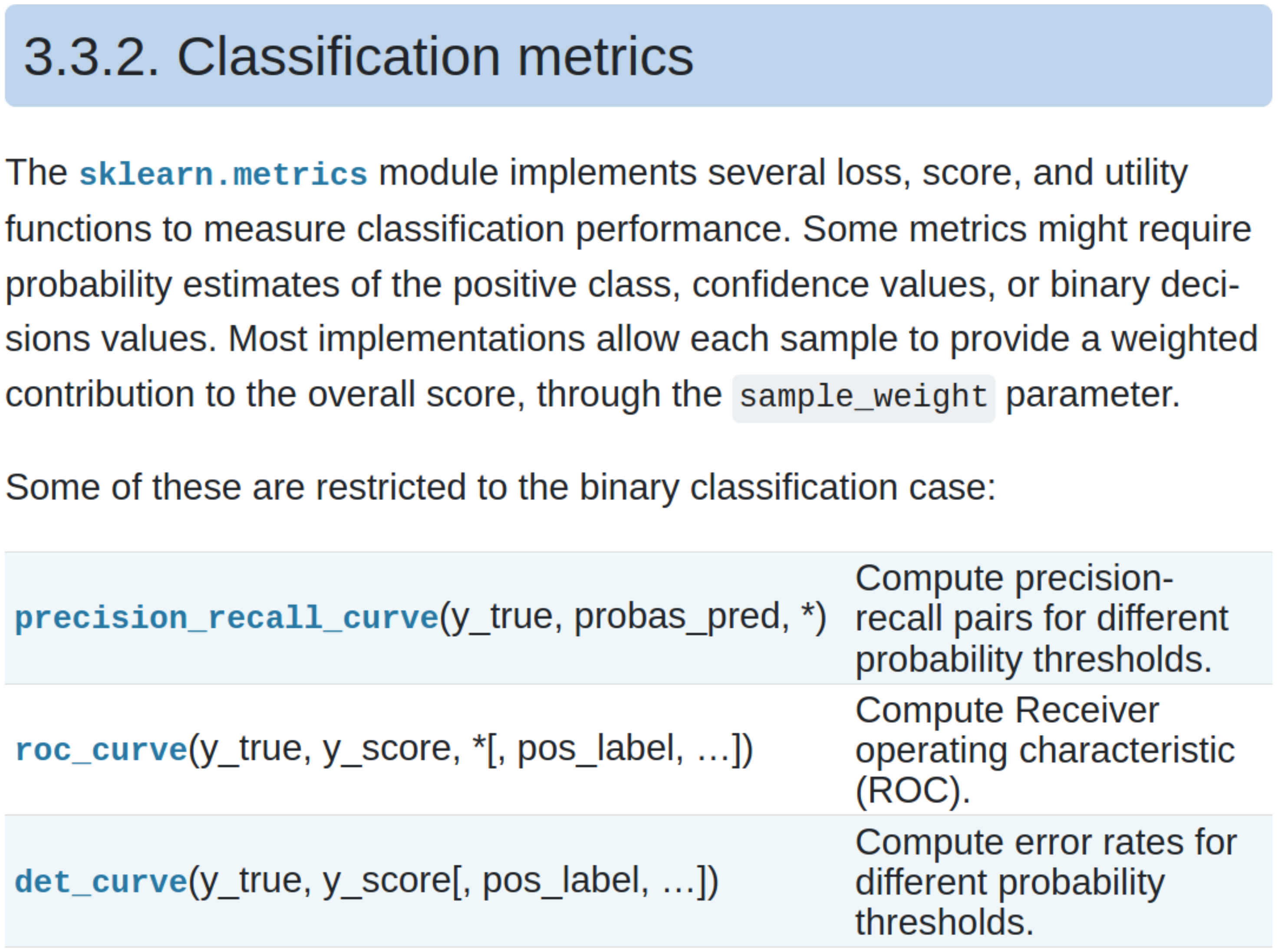
- By contrast, in **boosting methods**, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

  Examples: AdaBoost, Gradient Tree Boosting, …

Python

- **Scikit-Learn** package in Python: performance measures

https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics



3.3.2. Classification metrics

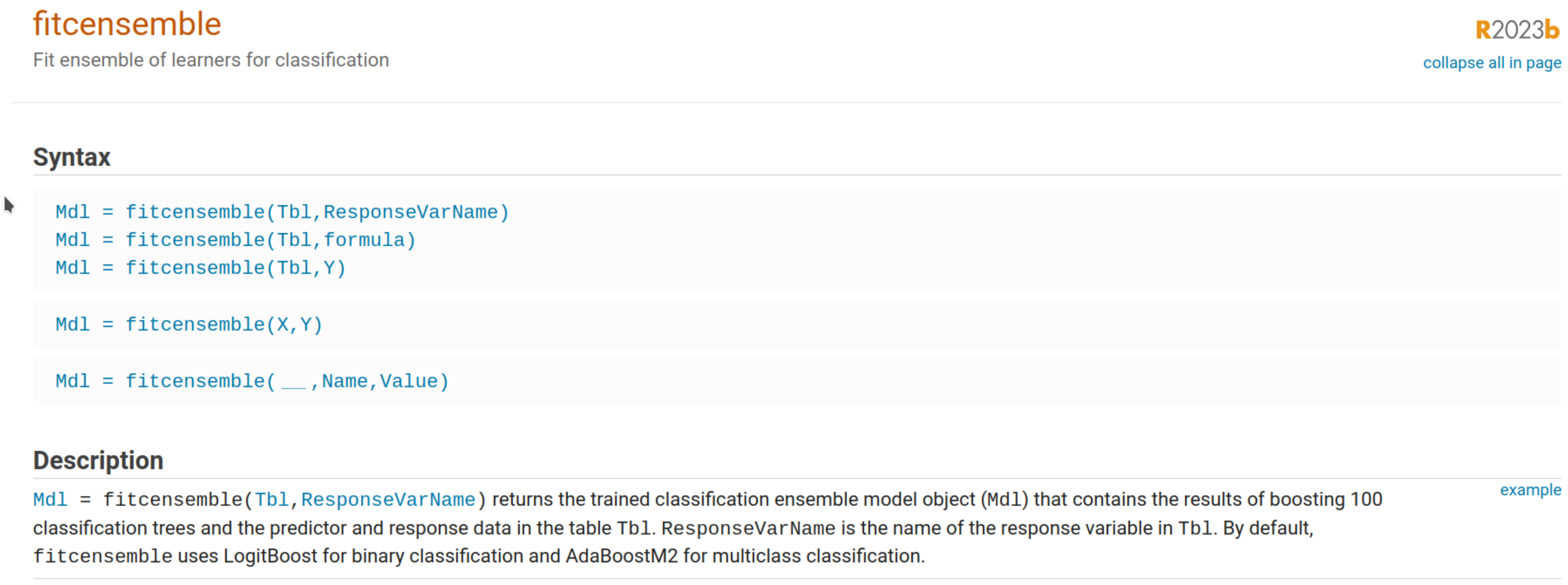
The **sklearn.metrics** module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values. Most implementations allow each sample to provide a weighted contribution to the overall score, through the `sample_weight` parameter.

Some of these are restricted to the binary classification case:

precision_recall_curve(y_true, probas_pred, *)	Compute precision-recall pairs for different probability thresholds.
roc_curve(y_true, y_score, *[, pos_label, …])	Compute Receiver operating characteristic (ROC).
det_curve(y_true, y_score[, pos_label, …])	Compute error rates for different probability thresholds.

Matlab

- **Statistics and Machine Learning Toolbox,** function **fitcensemble:**
  https://es.mathworks.com/help/stats/fitcensemble.html#d126e394981



fitcensemble

Fit ensemble of learners for classification

R2023b

collapse all in page

Syntax

```
Mdl = fitcensemble(Tbl,ResponseVarName)
Mdl = fitcensemble(Tbl,formula)
Mdl = fitcensemble(Tbl,Y)

Mdl = fitcensemble(X,Y)

Mdl = fitcensemble(___,Name,Value)
```

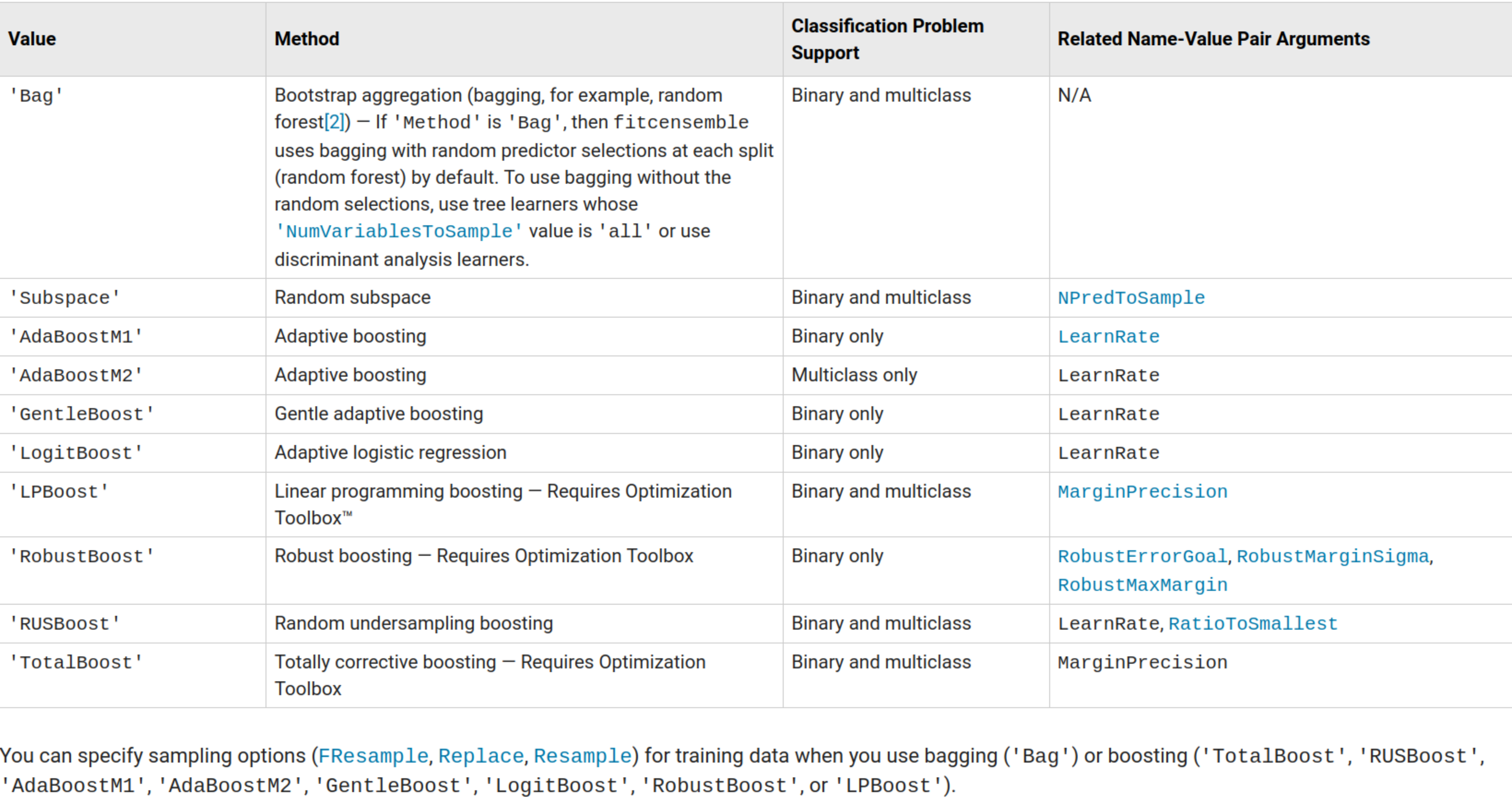
Description

example

`Mdl = fitcensemble(Tbl,ResponseVarName)` returns the trained classification ensemble model object (`Mdl`) that contains the results of boosting 100 classification trees and the predictor and response data in the table `Tbl`. `ResponseVarName` is the name of the response variable in `Tbl`. By default, `fitcensemble` uses LogitBoost for binary classification and AdaBoostM2 for multiclass classification.

Matlab

- **Statistics and Machine Learning Toolbox,** function **fitcensemble:** https://es.mathworks.com/help/stats/fitcensemble.html#d126e394981



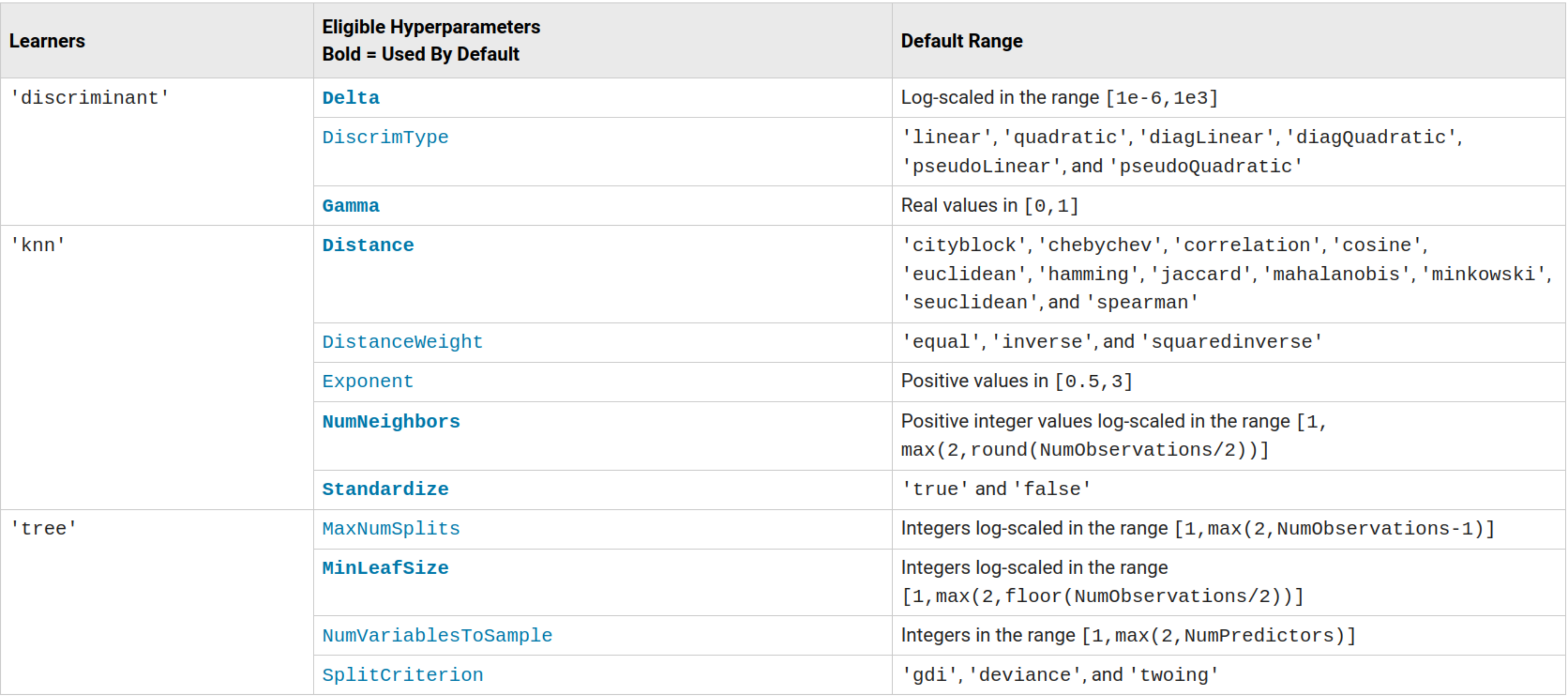
Value	Method	Classification Problem Support	Related Name-Value Pair Arguments
`'Bag'`	Bootstrap aggregation (bagging, for example, random forest[2]) — If `'Method'` is `'Bag'`, then `fitcensemble` uses bagging with random predictor selections at each split (random forest) by default. To use bagging without the random selections, use tree learners whose `'NumVariablesToSample'` value is `'all'` or use discriminant analysis learners.	Binary and multiclass	N/A
`'Subspace'`	Random subspace	Binary and multiclass	`NPredToSample`
`'AdaBoostM1'`	Adaptive boosting	Binary only	`LearnRate`
`'AdaBoostM2'`	Adaptive boosting	Multiclass only	`LearnRate`
`'GentleBoost'`	Gentle adaptive boosting	Binary only	`LearnRate`
`'LogitBoost'`	Adaptive logistic regression	Binary only	`LearnRate`
`'LPBoost'`	Linear programming boosting — Requires Optimization Toolbox™	Binary and multiclass	`MarginPrecision`
`'RobustBoost'`	Robust boosting — Requires Optimization Toolbox	Binary only	`RobustErrorGoal`, `RobustMarginSigma`, `RobustMaxMargin`
`'RUSBoost'`	Random undersampling boosting	Binary and multiclass	`LearnRate`, `RatioToSmallest`
`'TotalBoost'`	Totally corrective boosting — Requires Optimization Toolbox	Binary and multiclass	`MarginPrecision`

You can specify sampling options (`FResample`, `Replace`, `Resample`) for training data when you use bagging (`'Bag'`) or boosting (`'TotalBoost'`, `'RUSBoost'`, `'AdaBoostM1'`, `'AdaBoostM2'`, `'GentleBoost'`, `'LogitBoost'`, `'RobustBoost'`, or `'LPBoost'`).

Matlab: base classifiers

- **Statistics and Machine Learning Toolbox,** function **fitcensemble:**

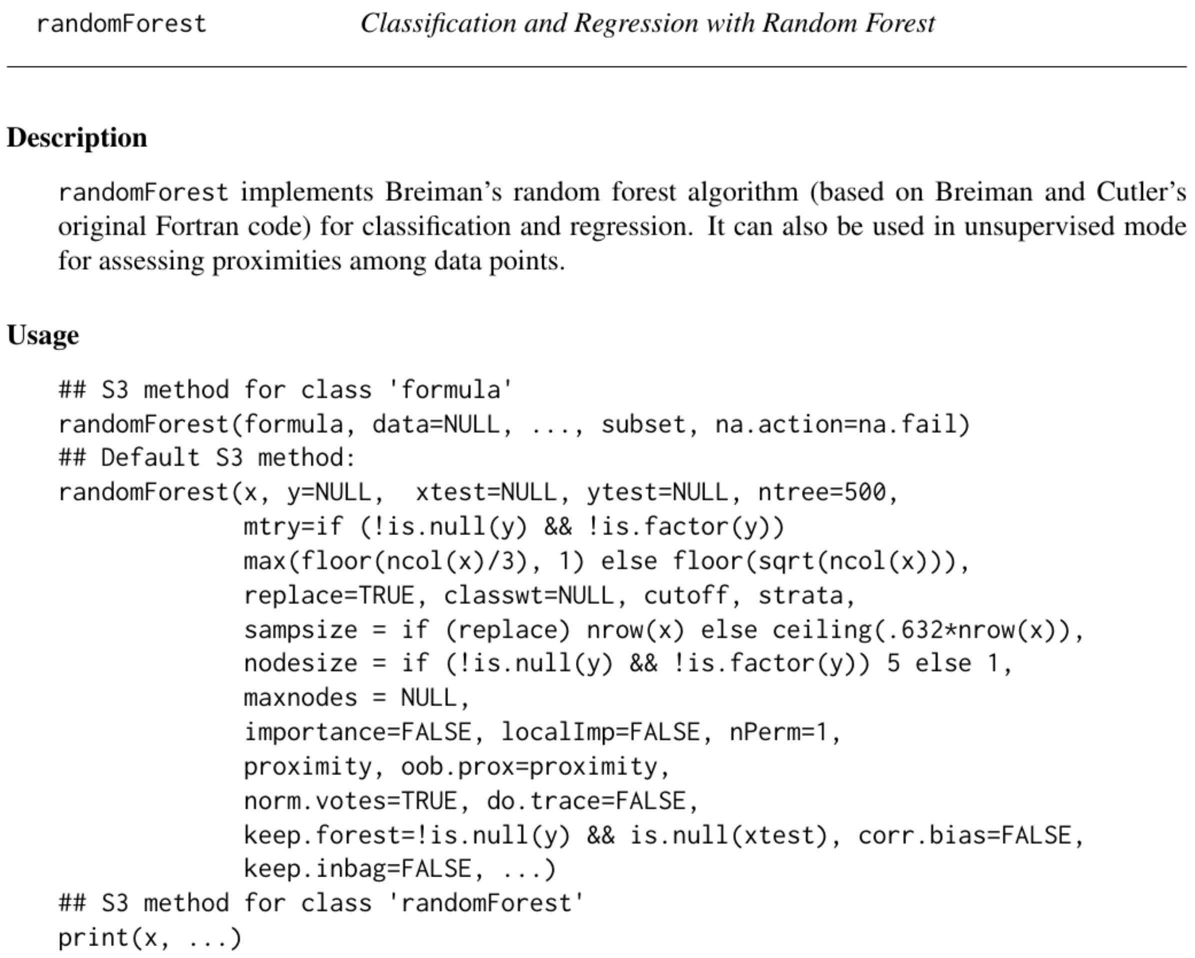
  https://es.mathworks.com/help/stats/fitcensemble.html#d126e394981



Learners	Eligible Hyperparameters Bold = Used By Default	Default Range
'discriminant'	**Delta**	Log-scaled in the range [1e-6,1e3]
	DiscrimType	'linear', 'quadratic', 'diagLinear', 'diagQuadratic', 'pseudoLinear', and 'pseudoQuadratic'
	Gamma	Real values in [0,1]
'knn'	**Distance**	'cityblock', 'chebychev', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', 'mahalanobis', 'minkowski', 'seuclidean', and 'spearman'
	DistanceWeight	'equal', 'inverse', and 'squaredinverse'
	Exponent	Positive values in [0.5,3]
	NumNeighbors	Positive integer values log-scaled in the range [1, max(2,round(NumObservations/2))]
	Standardize	'true' and 'false'
'tree'	MaxNumSplits	Integers log-scaled in the range [1,max(2,NumObservations-1)]
	MinLeafSize	Integers log-scaled in the range [1,max(2,floor(NumObservations/2))]
	NumVariablesToSample	Integers in the range [1,max(2,NumPredictors)]
	SplitCriterion	'gdi', 'deviance', and 'twoing'

R statistical computing language

- **RandomForest** package:



`randomForest` *Classification and Regression with Random Forest*

Description

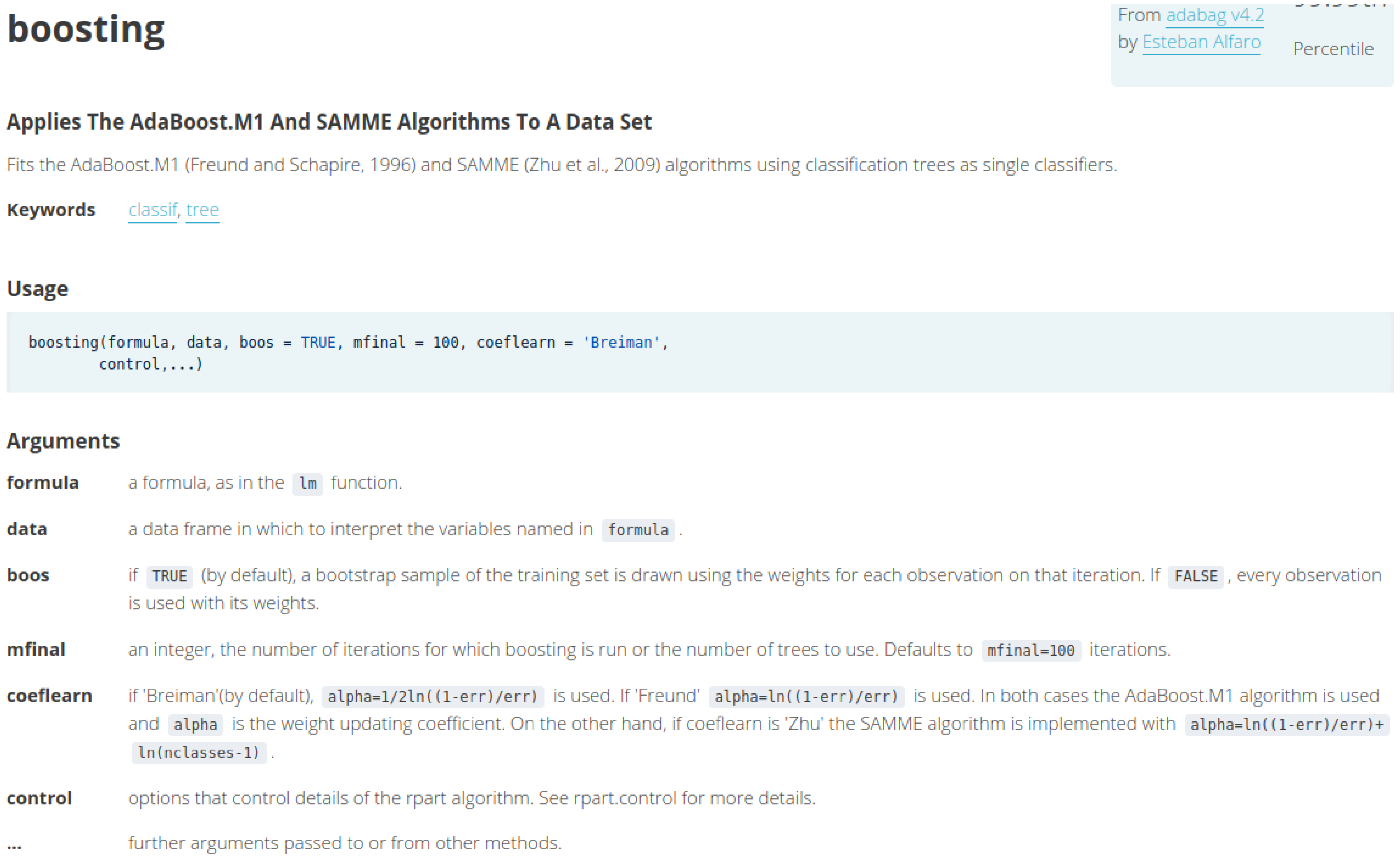
`randomForest` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

Usage

```
## S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
## S3 method for class 'randomForest'
print(x, ...)
```

R

- **adaBag** package:



boosting

From adabag v4.2
by Esteban Alfaro

Percentile

Applies The AdaBoost.M1 And SAMME Algorithms To A Data Set

Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.

Keywords classif, tree

Usage

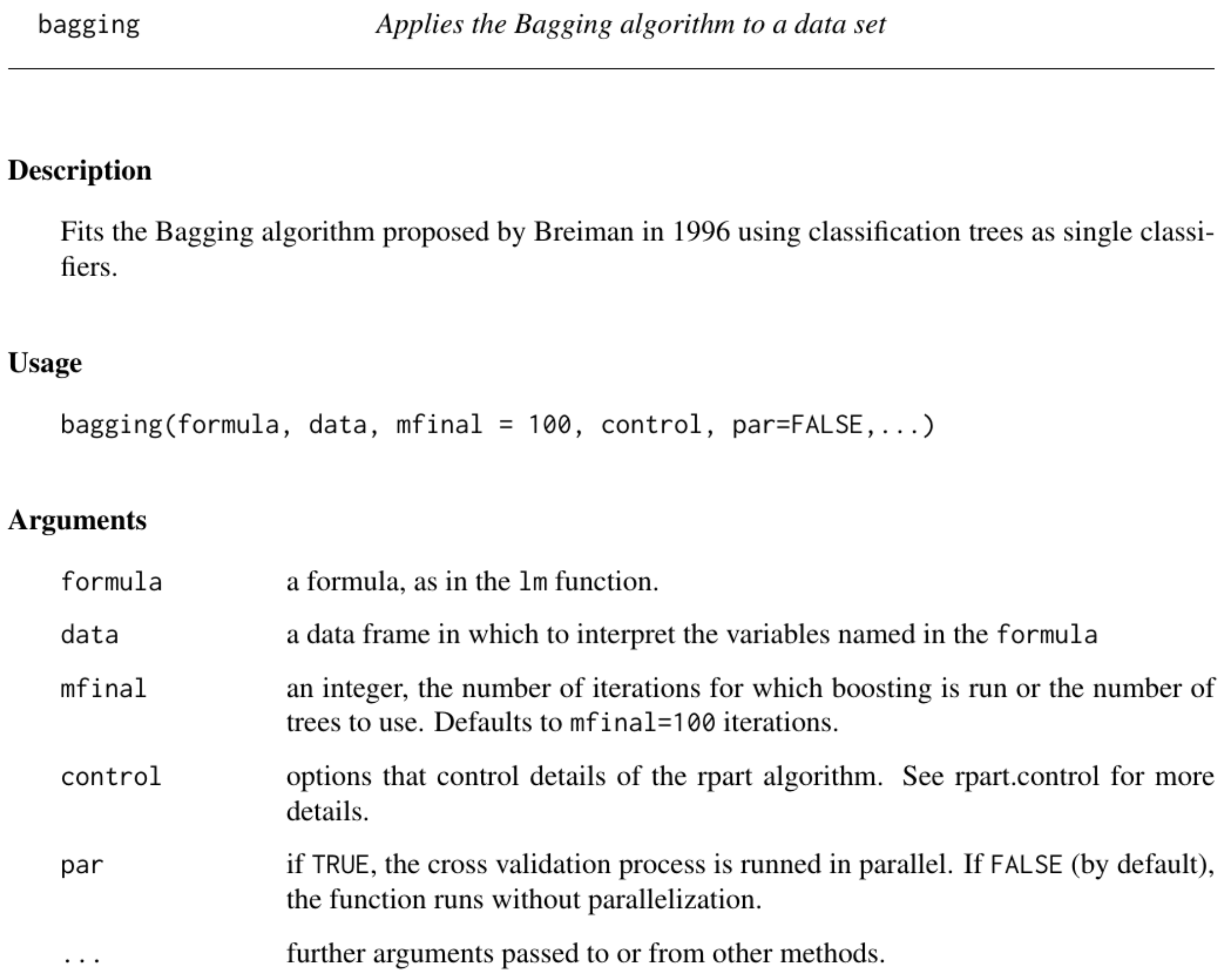
```
boosting(formula, data, boos = TRUE, mfinal = 100, coeflearn = 'Breiman',
        control,...)
```

Arguments

formula	a formula, as in the `lm` function.
data	a data frame in which to interpret the variables named in `formula` .
boos	if `TRUE` (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If `FALSE` , every observation is used with its weights.
mfinal	an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to `mfinal=100` iterations.
coeflearn	if 'Breiman'(by default), `alpha=1/2ln((1-err)/err)` is used. If 'Freund' `alpha=ln((1-err)/err)` is used. In both cases the AdaBoost.M1 algorithm is used and `alpha` is the weight updating coefficient. On the other hand, if coeflearn is 'Zhu' the SAMME algorithm is implemented with `alpha=ln((1-err)/err)+ ln(nclasses-1)` .
control	options that control details of the rpart algorithm. See rpart.control for more details.
...	further arguments passed to or from other methods.

R

- **adaBag** package:



`bagging` *Applies the Bagging algorithm to a data set*

Description

Fits the Bagging algorithm proposed by Breiman in 1996 using classification trees as single classifiers.

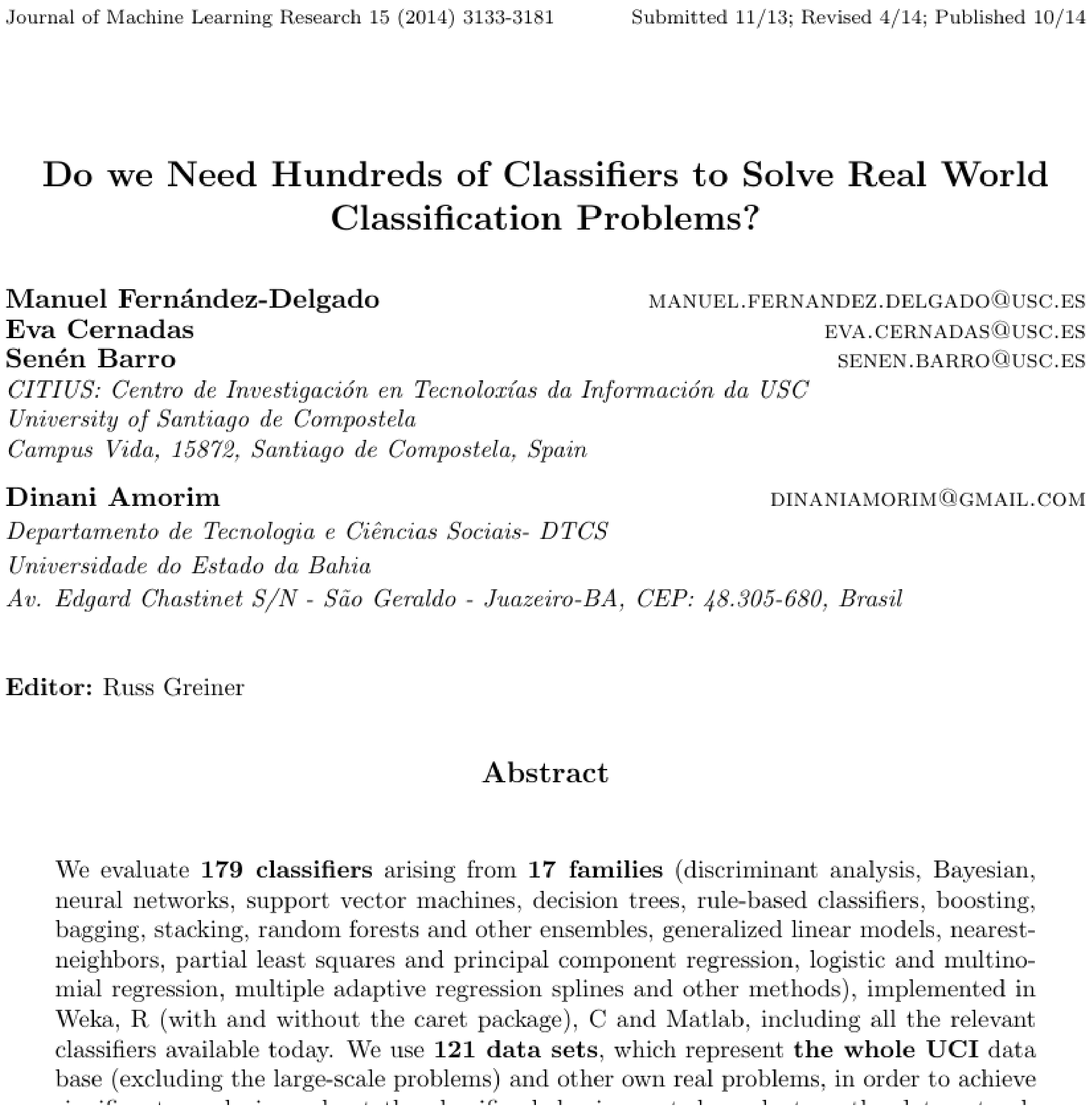
Usage

```
bagging(formula, data, mfinal = 100, control, par=FALSE,...)
```

Arguments

`formula`	a formula, as in the `lm` function.
`data`	a data frame in which to interpret the variables named in the `formula`
`mfinal`	an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to `mfinal=100` iterations.
`control`	options that control details of the rpart algorithm. See rpart.control for more details.
`par`	if `TRUE`, the cross validation process is runned in parallel. If `FALSE` (by default), the function runs without parallelization.
`...`	further arguments passed to or from other methods.

Classifiers comparison



Journal of Machine Learning Research 15 (2014) 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

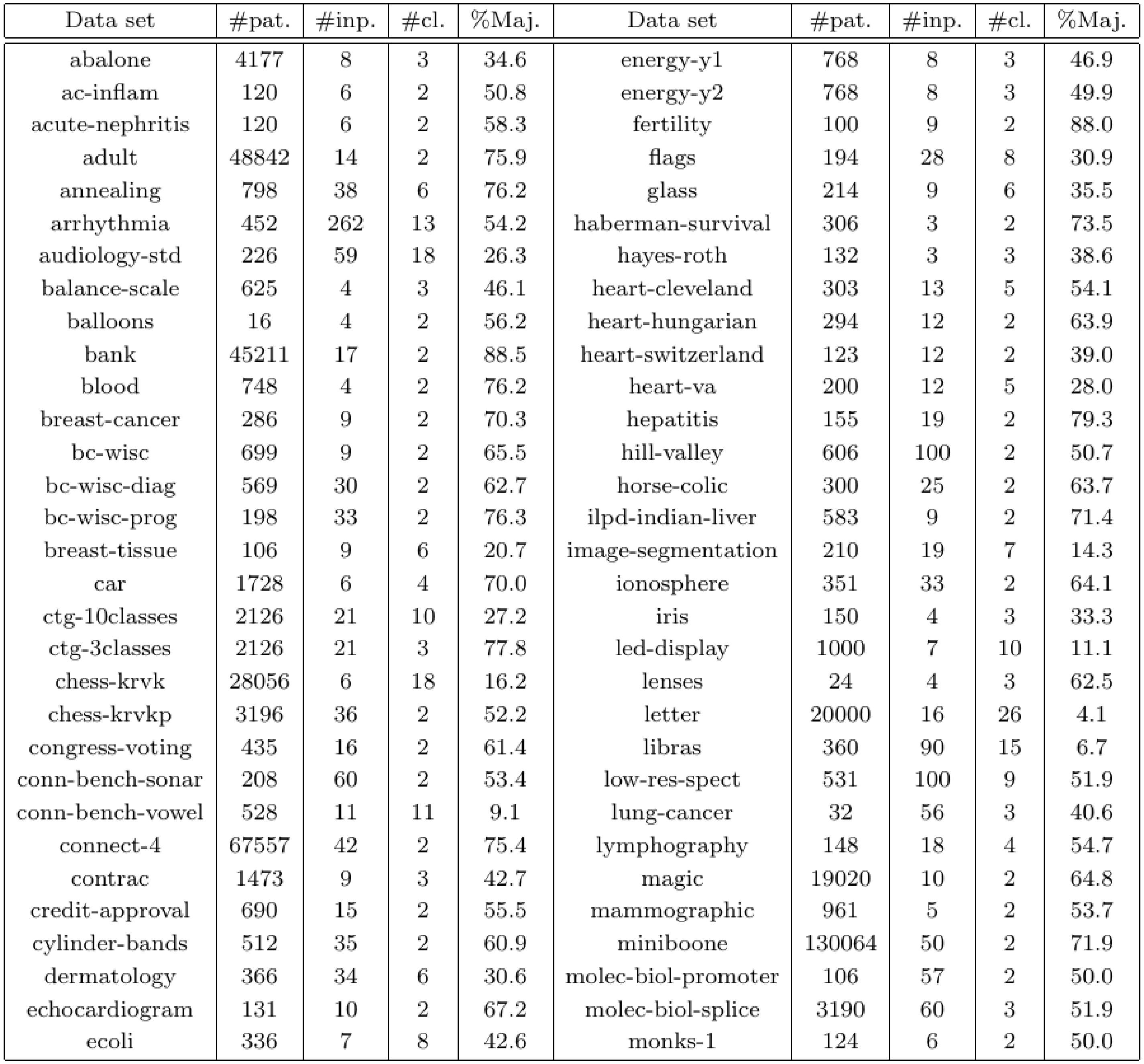
Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

Abstract

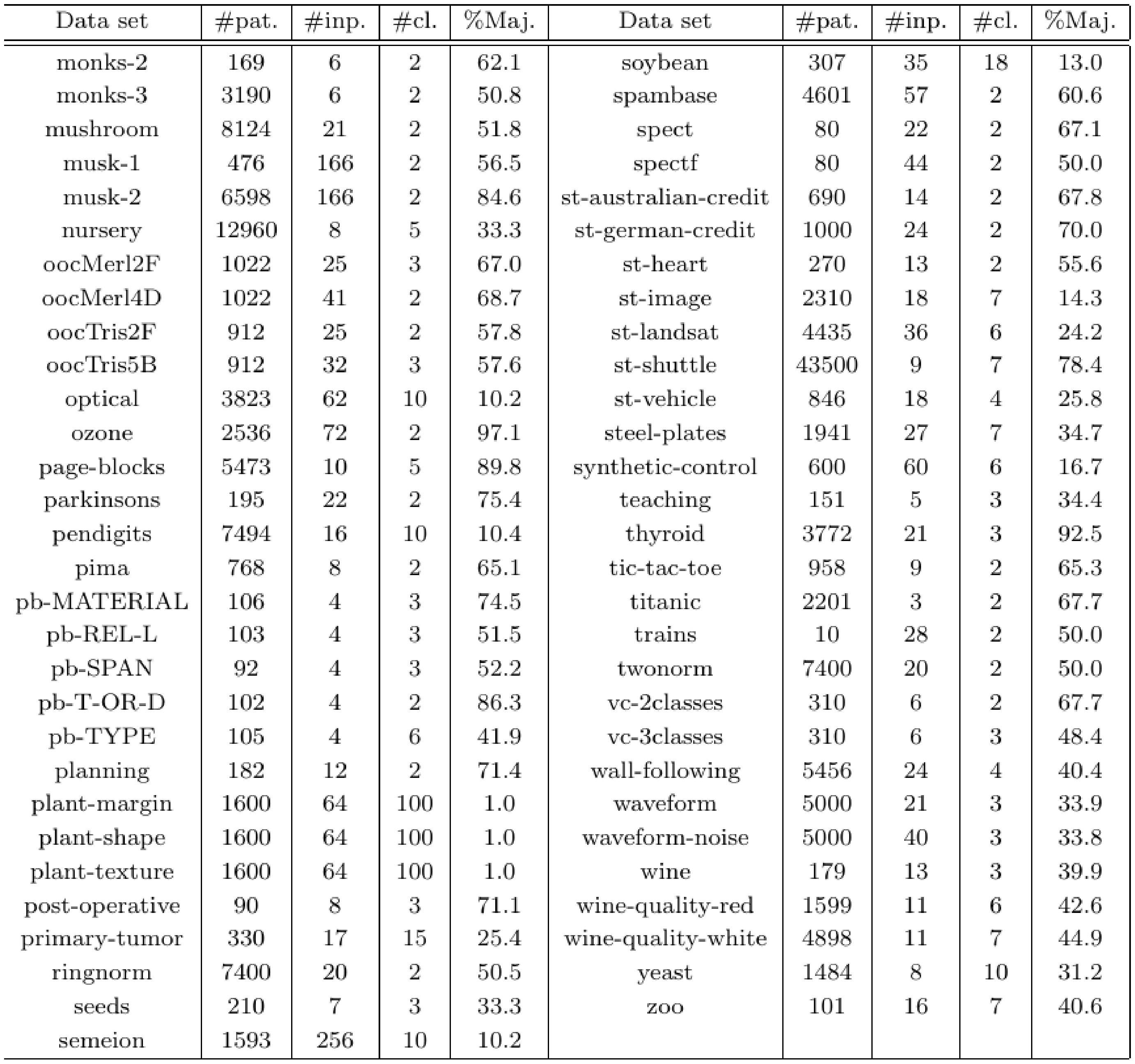
We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve

Classifiers comparison: datasets



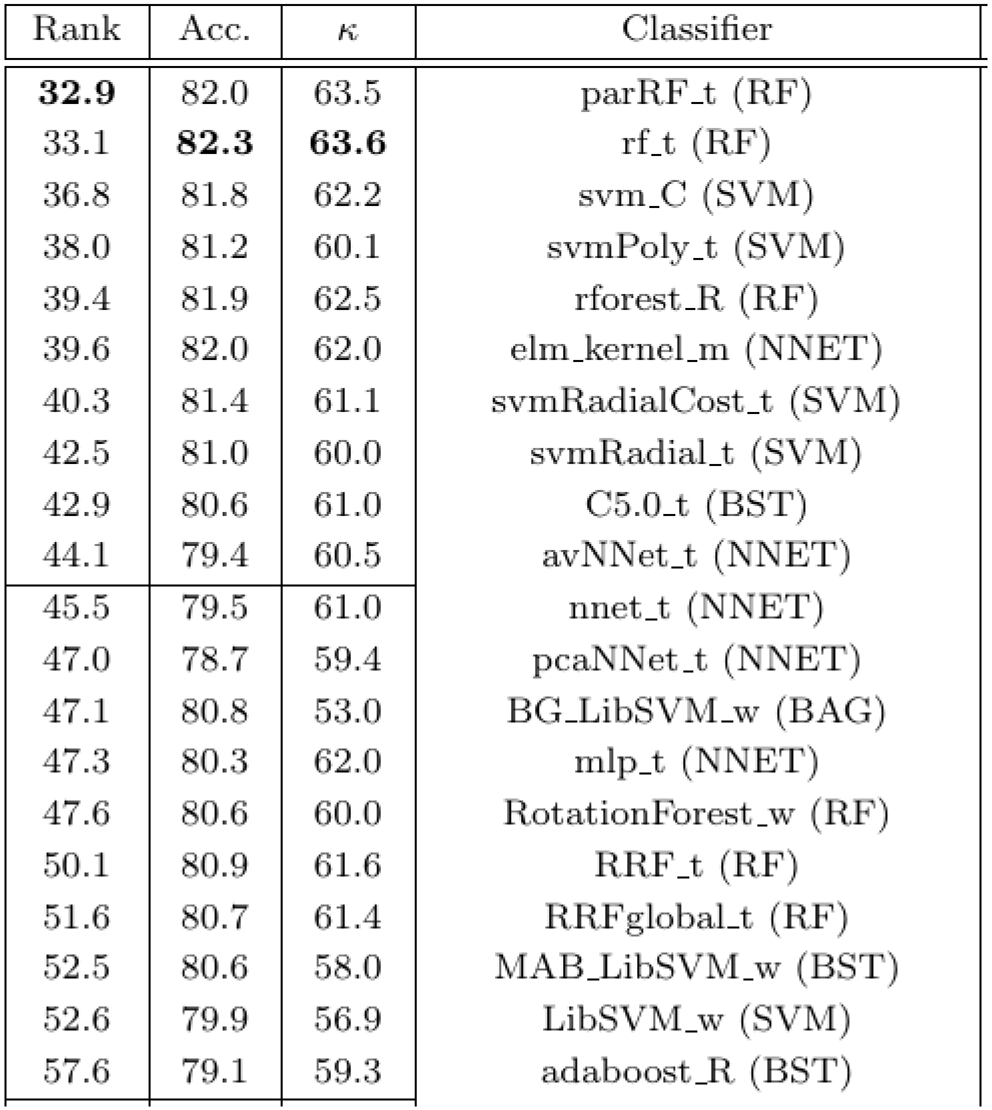
Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
abalone	4177	8	3	34.6	energy-y1	768	8	3	46.9
ac-inflam	120	6	2	50.8	energy-y2	768	8	3	49.9
acute-nephritis	120	6	2	58.3	fertility	100	9	2	88.0
adult	48842	14	2	75.9	flags	194	28	8	30.9
annealing	798	38	6	76.2	glass	214	9	6	35.5
arrhythmia	452	262	13	54.2	haberman-survival	306	3	2	73.5
audiology-std	226	59	18	26.3	hayes-roth	132	3	3	38.6
balance-scale	625	4	3	46.1	heart-cleveland	303	13	5	54.1
balloons	16	4	2	56.2	heart-hungarian	294	12	2	63.9
bank	45211	17	2	88.5	heart-switzerland	123	12	2	39.0
blood	748	4	2	76.2	heart-va	200	12	5	28.0
breast-cancer	286	9	2	70.3	hepatitis	155	19	2	79.3
bc-wisc	699	9	2	65.5	hill-valley	606	100	2	50.7
bc-wisc-diag	569	30	2	62.7	horse-colic	300	25	2	63.7
bc-wisc-prog	198	33	2	76.3	ilpd-indian-liver	583	9	2	71.4
breast-tissue	106	9	6	20.7	image-segmentation	210	19	7	14.3
car	1728	6	4	70.0	ionosphere	351	33	2	64.1
ctg-10classes	2126	21	10	27.2	iris	150	4	3	33.3
ctg-3classes	2126	21	3	77.8	led-display	1000	7	10	11.1
chess-krvk	28056	6	18	16.2	lenses	24	4	3	62.5
chess-krvkp	3196	36	2	52.2	letter	20000	16	26	4.1
congress-voting	435	16	2	61.4	libras	360	90	15	6.7
conn-bench-sonar	208	60	2	53.4	low-res-spect	531	100	9	51.9
conn-bench-vowel	528	11	11	9.1	lung-cancer	32	56	3	40.6
connect-4	67557	42	2	75.4	lymphography	148	18	4	54.7
contrac	1473	9	3	42.7	magic	19020	10	2	64.8
credit-approval	690	15	2	55.5	mammographic	961	5	2	53.7
cylinder-bands	512	35	2	60.9	miniboone	130064	50	2	71.9
dermatology	366	34	6	30.6	molec-biol-promoter	106	57	2	50.0
echocardiogram	131	10	2	67.2	molec-biol-splice	3190	60	3	51.9
ecoli	336	7	8	42.6	monks-1	124	6	2	50.0

Classifiers comparison: datasets



Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
monks-2	169	6	2	62.1	soybean	307	35	18	13.0
monks-3	3190	6	2	50.8	spambase	4601	57	2	60.6
mushroom	8124	21	2	51.8	spect	80	22	2	67.1
musk-1	476	166	2	56.5	spectf	80	44	2	50.0
musk-2	6598	166	2	84.6	st-australian-credit	690	14	2	67.8
nursery	12960	8	5	33.3	st-german-credit	1000	24	2	70.0
oocMerl2F	1022	25	3	67.0	st-heart	270	13	2	55.6
oocMerl4D	1022	41	2	68.7	st-image	2310	18	7	14.3
oocTris2F	912	25	2	57.8	st-landsat	4435	36	6	24.2
oocTris5B	912	32	3	57.6	st-shuttle	43500	9	7	78.4
optical	3823	62	10	10.2	st-vehicle	846	18	4	25.8
ozone	2536	72	2	97.1	steel-plates	1941	27	7	34.7
page-blocks	5473	10	5	89.8	synthetic-control	600	60	6	16.7
parkinsons	195	22	2	75.4	teaching	151	5	3	34.4
pendigits	7494	16	10	10.4	thyroid	3772	21	3	92.5
pima	768	8	2	65.1	tic-tac-toe	958	9	2	65.3
pb-MATERIAL	106	4	3	74.5	titanic	2201	3	2	67.7
pb-REL-L	103	4	3	51.5	trains	10	28	2	50.0
pb-SPAN	92	4	3	52.2	twonorm	7400	20	2	50.0
pb-T-OR-D	102	4	2	86.3	vc-2classes	310	6	2	67.7
pb-TYPE	105	4	6	41.9	vc-3classes	310	6	3	48.4
planning	182	12	2	71.4	wall-following	5456	24	4	40.4
plant-margin	1600	64	100	1.0	waveform	5000	21	3	33.9
plant-shape	1600	64	100	1.0	waveform-noise	5000	40	3	33.8
plant-texture	1600	64	100	1.0	wine	179	13	3	39.9
post-operative	90	8	3	71.1	wine-quality-red	1599	11	6	42.6
primary-tumor	330	17	15	25.4	wine-quality-white	4898	11	7	44.9
ringnorm	7400	20	2	50.5	yeast	1484	8	10	31.2
seeds	210	7	3	33.3	zoo	101	16	7	40.6
semeion	1593	256	10	10.2					

Classifiers comparison: Friedman rank



Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)
33.1	**82.3**	**63.6**	rf_t (RF)
36.8	81.8	62.2	svm_C (SVM)
38.0	81.2	60.1	svmPoly_t (SVM)
39.4	81.9	62.5	rforest_R (RF)
39.6	82.0	62.0	elm_kernel_m (NNET)
40.3	81.4	61.1	svmRadialCost_t (SVM)
42.5	81.0	60.0	svmRadial_t (SVM)
42.9	80.6	61.0	C5.0_t (BST)
44.1	79.4	60.5	avNNet_t (NNET)
45.5	79.5	61.0	nnet_t (NNET)
47.0	78.7	59.4	pcaNNet_t (NNET)
47.1	80.8	53.0	BG_LibSVM_w (BAG)
47.3	80.3	62.0	mlp_t (NNET)
47.6	80.6	60.0	RotationForest_w (RF)
50.1	80.9	61.6	RRF_t (RF)
51.6	80.7	61.4	RRFglobal_t (RF)
52.5	80.6	58.0	MAB_LibSVM_w (BST)
52.6	79.9	56.9	LibSVM_w (SVM)
57.6	79.1	59.3	adaboost_R (BST)

Classifiers comparison: conclusions

- Random Forest (RF) and Support Vector Machine (SVM) families (with different implementations and approaches) are the strongest classifiers (achieved the first positions in the Friedman rank).
- High performance also for Extreme Learning Machine (ELM) (6th position).
- Other neural networks (NNET), boosting (BST) and bagging (BAG) classifiers achieved also good performance.