

Package ‘ProbSVMs’

September 13, 2026

Type Package

Title Probabilistic Support Vector Machines

Version 0.2.0

Author A. Pedro Duarte Silva [aut, cre]

Maintainer A. Pedro Duarte Silva <psilva@ucp.pt>

Description Implements kernel-based classification Support Vector Machines with reliable estimated probabilities of class membership. Theoretical support for the functions in this package can be found in Duarte Silva (2025) <[doi:10.1016/j.cor.2025.107203](https://doi.org/10.1016/j.cor.2025.107203)>.

License GPL-2

Encoding UTF-8

Depends R (>= 3.5.0), Rcpp (>= 1.0.4.6)

LinkingTo Rcpp, RcppArmadillo (>= 15.2.6-1)

Imports lpSolveAPI, kernlab, ggplot2, MASS, stats, reshape2

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-09-12 23:50:02 UTC

Contents

ProbSVMs-package	2
as.ClassProb	4
GetrbfdotSigPar	5
makeKMat	6
plot.ClassProb	7
predict.kernelSVM	10
predict.PVM	11
SetUpOptPar	14
SetUpTunPar	15
trainPVM	16
trainSVM	21

Index	28
--------------	-----------

 ProbSVMs-package

Probabilistic Support Vector Machines

Description

Implements 'all-in-one' kernel-based multiclass Support Vector Machines for supervised classification and class probability estimation.

Details

Package:	ProbSVMs
Type:	Package
Version:	0.2-0
Date:	2026-09-13
License:	GPL-2
LazyLoad:	yes

Package **ProbSVMs** implements 'all-in-one' kernel-based multiclass Support Vector Machines for supervised classification and class probability estimation. The most original novelty of **ProbSVMs** is the Probabilistic Vector Machines (PVMs) proposed by Duarte Silva in reference [3]. PVMs are distribution-free estimators of class probabilities based on the predictions given by a sequence of kernel-based weighted Supported Vector Machines (SVMs).

Currently there two variants of multiclass PVMs implemented in **ProbSVMs**: (i) machines that use the SVM loss proposed by Lee, Lin and Wahba (see reference [5]), and machine that use the loss proposed by Weston and Watkins (see reference [6]). The former variant has better asymptotic properties, but there are some empirical evidence suggesting that the later often gives more reliable results in many applications (see, e.g., reference [2]). For two-class problems these two variants are equivalent.

In problems with more than two classes currently **ProbSVMs** only implements PVMs based on classification rules derived from weighted kernel-based SVMs without bias terms. In two-class problems both rules with or without (default) bias terms can be used. It can be shown that dropping bias terms in kernel-based SVMs has computational advantages, and it has been argued that it should not have a noticable impact on statistical performance (see references [2] and [3] for further discussion).

In addition, **ProbSVMs** also implements Duarte Silva's adaptation (see reference [3]) of Dogan, Glasmachers and Igel' s efficient training algorithm (see reference [1]) for 'all-in-one' multiclass kernel-based SVMs without bias terms. While an original implementation of this algorithm is available on the machine learning Shark library (see reference [4]), unlike **ProbSVMs** that implementation does not seems to cover weighted SVM versions, nor does it seem to include any interface to the R environment.

in **ProbSVMs**, PVMs can be trained by the function `trainPVM` which generates an object of class PVM, which has a predict method for computing reliable class probability estimates. Similarly, kernel-based SVMs can be trained by the function `trainSVM` which generates an object of class `kernelSVM` (single trained SVM) or `kernelSVMs` (multiple trained SVMs) that have predict methods for finding class predictions.

Author(s)

Antonio Pedro Duarte Silva <psilva@ucp.pt>

Maintainer: Antonio Pedro Duarte Silva <psilva@ucp.pt>

References

- [1] Dogan, U.; Glasmachers, T. and Igel, C. (2011) Fast training of multi-class support vector machines. *Technical report. Department of Computer Sciences, University of Copenhagen.*
- [2] Dogan U.; Glasmachers T. and Igel, C. (2016) A unified view on multi-class support vector classification. *Journal of Machine Learning Research*, Vol. 17 (45), 1–32.
- [3] Duarte Silva, A.P. (2025) Probabilistic Vector Machines. *Computers and Operations Research*, Vol. 183, 107203. <doi:10.1016/j.cor.2025.107203>
- [4] Igel, C.; Heidrich-Meisner, V. and Glasmachers, T. (2008) Shark. *Journal of Machine Learning Research*, Vol 9 (6), 993-996.
- [5] Lee, Y.; Lin, Y. and Wahba, G. (2004) Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, Vol. 99, 67–81. <doi:10.1198/016214504000000098>
- [6] Weston, J. and Watkins, C. (1999) Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 219–224.

See Also

[trainPVM](#), [trainSVM](#), [predict.PVM](#), [predict.kernelSVM](#), [plot.ClassProb](#)

Examples

```
# train the Weston and Watkins SVM on the Iris data

data(iris)
WWsvm <- trainSVM(iris[,1:4],iris$Species,loss="WW")

# Get in-sample classification results

WWpred <- predict(WWsvm,newdata=iris[,1:4])
WWpred

# Compare classifications with true assignments

cat("Original classes:\n")
print(iris$Species)
print(WWpred==iris$Species)

# Estimate class probabilities based on the WW loss

set.seed(1) # Fix the seed in order to get reproducible results
# (given the the building ot the grid matrix has a random component)
```

```

WWpvm <- trainPVM(iris[,1:4],iris$Species,loss="WW")
WWprob <- predict(WWpvm,newdata=iris[,1:4],trndt="newdata")

# Display the probabilities of the predicted classes
# on the space of the petal measurements

plot(WWprob, projecton="OrigDt", trdata=iris, axis=c(3,4))

```

as.ClassProb

Coerce matrix and data.frames to a ClassProb object

Description

conversion function to create ClassProb objects form matrices and data.frames.

Usage

```
as.ClassProb(x, ...)
```

Arguments

x	a matrix or data frame (with observations in rows and classes in columns) containing class probability estimates in a supervised classification problem.
...	further arguments passed to or from other methods.

Value

an object of class ClassProb containing class probability estimates, for which there is a specialized plot method. The internal structure of objects inheriting from class ClassProb is simply the original matrix or data frame plus its class attribute.

See Also

[plot.ClassProb](#)

GetrbfdotSigPar	<i>rbfdot Sigma hyper-parameter</i>
-----------------	-------------------------------------

Description

GetrbfdotSigPar returns sensible values for the sigma parameter of the rbfdot (“Gaussian”) kernel function.

Usage

```
GetrbfdotSigPar(x, kpar=c("d2median", "d2q01q09mean", "d2q01q09hmean"))
```

Arguments

- | | |
|------|--|
| x | a matrix or data frame of training data (observations in rows, variables in columns). |
| kpar | a string specifying how sigma should be computed. Valid alternatives are: <ul style="list-style-type: none"> • “d2median”: the median of the the euclidean distances between all observations pairs in the data given by argument x. • “d2q01q09mean”: the arithmetic mean of the 10-th and 90-th percentiles of the the euclidean distances between all observations pairs in the data given by argument x. • “d2q01q09hmean”: the harmonic mean of the 10-th and 90-th percentiles of the the euclidean distances between all observations pairs in the data given by argument x. |

Details

GetrbfdotSigPar returns a sensible value for the sigma parameter of the Radial Basis (“Gaussian”) kernel function. According to Caputo et. al (reference [1]) any sigma values between the 10-th and 90-th percentiles of the distribution of the pairwise euclidean distances between observation pairs, $d_{ij}^2 = ||x_i - x_j||^2$, lead to reasonable Support Vector Machine classification results. GetrbfdotSigPar returns one of three alternatives for the sigma value: (i) the median of the d_{ij}^2 distribution (default). (ii) the arithmetic mean of the 10-th and 90-th percentiles of the d_{ij}^2 distribution. (iii) the harmonic mean of the 10-th and 90-th percentiles of the d_{ij}^2 distribution.

Value

a scalar with a sensible value for sigma hyper-parameter of the Radial Basis (“Gaussian”) kernel function.

References

[1] Caputo, B.; Sim, K.; Furesjo, F. and Smola, A. (2002). Appearance-based object recognition using svms: which kernel should i use? In *Proceedings of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision*, Whistler.

See Also

[trainPVM](#), [trainSVM](#).

makeKMat

making kernel matrices

Description

makeKMat creates kernel matrices from row data. It can be used to create the Gram matrix from a single data set, or a general kernel matrix from two different data sets.

Usage

```
makeKMat(dt1, dt2=NULL, kernel=c("rbfdot", "vanilladot", "polydot"),
         kpar=list(sigma="d2median"))
```

Arguments

- | | |
|--------|---|
| dt1 | a data matrix or data frame, with observations in rows and variables in columns. |
| dt2 | a data matrix or data frame, with observations in rows and variables in columns. If equal to NULL the kernel matrix to be created will consist of kernel values between all pairs of the dt1 elements. Otherwise, it will consist of the kernel values between the elements of dt1 (in rows) and dt2 (in columns). |
| kernel | the kernel function used for training and prediction. Currently this argument can be set to one of the following strings: <ul style="list-style-type: none"> • rbfdot Radial Basis kernel “Gaussian” • vanilladot Linear kernel • polydot Polynomial kernel |
| kpar | the list of kernel hyper-parameters. This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : <ul style="list-style-type: none"> • sigma: inverse kernel width for the Radial Basis kernel function “rbfdot”. Either a numeric value or one of the strings “d2median” (default), “d2q01q09mean”, or “d2q01q09hmean”. In the later case this value is computed automatically, from the distribution of all euclidean distances between dt1 observation pairs. See GetrbfdotSigPar for details. • degree, scale, offset: for the Polynomial kernel “polydot” |

Value

A kernel matrix. If argument dt2 is NULL, a symmetric matrix consisting of the kernel values between all pairs of dt1 elements. If argument dt2 is not NULL, a matrix with the kernel values between the elements of the dt1 (in rows) and dt2 (in columns).

See Also

[GetrbfdotSigPar](#)

plot.ClassProb

*S3 plot method for ClassProb objects***Description**

Method for displaying the class probabilities contained in ClassProb objects.

Usage

```
## S3 method for class 'ClassProb'
plot(x, ..., type=c("scatterplt", "stckdbarplt"),
     projecton=c("DiscFact", "PCs", "OrigDt"), trdata=NULL, grouping=NULL,
     newdata="trdata", ShownCl="PredCl", axis=c(1,2), obs=1:nrow(x),
     withxlabels=length(obs)<=30, title=NULL, pntsize = 2.5, threecolors=TRUE,
     clgrdparl = list(low="white", mid="blue", high="darkred", midpoint=0.7))
```

Arguments

x	an object of class inheriting from “ClassProb”.
...	further arguments passed to or from other methods.
type	type of plot to be displayed. Currently this argument can be set to one of the following strings: • “scatterplt” scatter plot of projections on a space defined by the value of argument projecton. Probabilities are represented by different colors according to the colour scale defined by the value of arguments threecolors and clgrdparl. • “stckdbarplt” stacked bar plot. Probabilites are represented by the size of different colour groups.
projecton	Type of projection used to display the class probabilities. The value of this argument is ignored when argument type is equal to “stckdbarplt”. Currently this argument can be set to one of the following strings: • “DiscFact” Discriminant space formed by two (problems with three o more classes) or the unique (problems with two classes) canonical Linear Discriminant Factors (LDFs). With problems with more than three groups the set of LDFs chosen is determined by the value of argument axis. By default the first two LDFs are chosen. • “PCs” Principal space formed by two Principal Components. The set of PCs chosen is determined by the value of argument axis. By default the first two PCs are chosen. • “OrigDt” Space of the original variables corresponding to trdata columns determined by the value of argument axis.
trdata	either a matrix or data frame with the training data (with observations in rows and variables in columns) based on which the scatter plot projections are to be found. The value of this argument is ignored when argument type is equal to “stckdbarplt”.

grouping	a factor describing the response vector, with one label by observation, in the training data. This argument is only required for projections onto canonical Linear Discriminant spaces, and its value is ignored when argument <code>argument type</code> is equal to “ <code>stckdbarplt</code> ”, or argument <code>projecton</code> is different from “ <code>DiscFact</code> ”.
newdata	either a matrix or data frame with the data (with observations in rows and variables in columns) for which class probabilities are to be displayed.
ShownCl	a description of the class whose probabilities are to be displayed by a colour grid in the scatter plot projection displays. This could be either the string “ <code>PredCl</code> ” (default), if for each observation the probabilities of the predicted class are to be displayed, or the position or name of a particular class. The value of this argument is ignored when argument <code>type</code> is equal to “ <code>stckdbarplt</code> ”.
axis	the set of LDFs or PCs (when argument <code>projecton</code> is equal to “ <code>DiscFact</code> ” or “ <code>PCs</code> ”) or column positions (when argument <code>projecton</code> is equal to “ <code>OrigDt</code> ”) used to define the projections in scatter plot displays. The value of this argument is ignored when argument <code>type</code> is equal to “ <code>stckdbarplt</code> ”.
obs	the set of <code>newdata</code> observations for which class probabilities are to be displayed. Its value can be either equal to the row indices or row names of the data provided by argument <code>newdat</code> .
withxlabels	a logical flag indicating if, in stacked bar plot displays, the observation names are to be shown on the x axis. The value of this argument is ignored when argument <code>type</code> is equal to “ <code>scatterplt</code> ”.
title	an overall title for the plot.
pntsize	a numerical value giving the amount by which, in scatter plots, plotting symbols should be magnified relative to the base plot default. The value of this argument is ignored when argument <code>type</code> is equal to “ <code>stckdbarplt</code> ”.
threecolors	logical flag indicating if, in scatter plot displays, the colour scheme used for representing class probabilities should be based on a three-colour scale (TRUE), or a more conventional (but somehow less discriminatory) gradient colour scale (FALSE). The value of this argument is ignored when argument <code>type</code> is equal to “ <code>stckdbarplt</code> ”.
clgrdparl	a list of arguments used to define the colour scale used for representing class probabilities in scatter plot projections. The value of this argument is ignored when argument <code>type</code> is equal to “ <code>scatterplt</code> ”. The components of this list are : • <code>low</code>: the colour associated with low class probabilities. • <code>mid</code>: the colour associated with medium class probabilities. The value of this argument is ignored when argument <code>threecolors</code> is equal to FALSE. • <code>high</code>: the colour associated with high class probabilities. • <code>midpoint</code>: the medium class probability associated with the colour given by the <code>mid</code> component. The value of this argument is ignored when argument <code>threecolors</code> is equal to FALSE.

Value

No return value, called for side effects

See Also[trainPVM](#), [predict.PVM](#), [as.ClassProb](#)**Examples**

```
# Train the Weston and Watkins PVM on the Iris data
# and obtain the training sample class probabilities

data(iris)

WWpvm <- trainPVM(iris[,1:4],iris$Species)
WWprob <- predict(WWpvm,newdata=iris[,1:4],trndt="newdata")

# Display the probabilities of the predicted classes
# on the two-dimensional canonical discriminant space

plot(WWprob, trdata=iris[,1:4], grouping=iris$Species)

# Display the same probabilities on the space of the petal measurements

plot(WWprob, projecton="OrigDt", trdata=iris, axis=c(3,4))

# Show all class probabilities by a stacked bar plot

plot(WWprob, type="stckdbarplt")

# Display the scatter plots, focusing now on the virginica probabilities

plot(WWprob, trdata=iris[,1:4], grouping=iris$Species, ShownCl="virginica")
plot(WWprob, projecton="OrigDt", trdata=iris, axis=c(3,4), ShownCl="virginica")

# Repeat the previous analysis, using the first 40 observations in each class
# for training, and the last 10 for probability estimation

trdata <- iris[c(1:40,51:90,101:140),1:4]
trSpecies <- iris$Species[c(1:40,51:90,101:140)]
evaldata <- iris[c(41:50,91:100,141:150),1:4]
WWpvm1 <- trainPVM(trdata,trSpecies)
WWprob1 <- predict(WWpvm1,newdata=evaldata)

plot(WWprob1, projecton="DiscFact", trdata=trdata, newdata=evaldata, grouping=trSpecies)
plot(WWprob1, projecton="OrigDt", newdata=evaldata)
plot(WWprob1, type="stckdbarplt", title = "Estimated class probabilities")
plot(WWprob1, projecton="DiscFact", trdata=trdata, newdata=evaldata,
      grouping=trSpecies, ShownCl="virginica")
plot(WWprob1, projecton="OrigDt", newdata=evaldata, ShownCl="virginica")
```

predict.kernelSVM *S3 predict method for class 'kernelSVM'*

Description

predict classes based on object of class 'kernelSVM'.

Usage

```
## S3 method for class 'kernelSVM'
predict(object, ..., newdata=NULL, KMat=NULL, trndt=NULL)
```

Arguments

object	Object of class inheriting from “kernelSVM”. This could either contain a single trained SVM (if the object class is “kernelSVM”) or multiple trained SVMs (if the object class is “kernelSVMs”).
...	further arguments passed to or from other methods.
newdata	a matrix or data frame with the new data (with observations in rows and variables in columns) to be predicted by the SVM(s) given by object. It is ignored when argument KMat is not NULL.
KMat	a Kernel matrix with the values resulting from applying the kernel function to all pairs between newdata and training data observations. The KMat rows are associated with the new observations, while its columns correspond to training data observations. When not NULL overrides the newdata argument.
trndt	either a NULL value, a matrix or data frame with the training data (with observations in rows and variables in columns) or the string “newdata”, if (and only if) class probabilities are to be computed for the same data as the one used for training. When equal to NULL the training data will be retrieved from the value of argument object. The value of this argument is ignored when argument KMat is not NULL.

Value

For predictions based on single SVM objects (classe “kernelSVM”), a factor with the class predictions for each new data observation. For predictions based on multiple SVM objects (class “kernelSVMs”), a data frame of factors where each column corresponds to a different SVM and each row corresponds to a new data observation.

See Also

[trainSVM](#)

Examples

```
## The following examples are based on the MASS data set "crabs".
# This data records physical measurements on 200 specimens of
# Leptograpsus variegatus crabs observed on the shores
## of Western Australia. The crabs are classified by two factors, sex and sp
## (crab species as defined by its colour: blue or orange), with two levels
## each. The measurement variables include the natural logarithms of carapace length (CL),
## the carapace width (CW), the size of the frontal lobe (FL) and the size of
## the rear width (RW). In the analysis provided, we created four classes
# by crossing the sex and sp levels.

library(MASS)
data(crabs)

crabs$grp <- interaction(crabs$sex,crabs$sp)

crabs$lnFL <- log(crabs$FL)
crabs$lnRW <- log(crabs$RW)
crabs$lnCL <- log(crabs$CL)
crabs$lnCW <- log(crabs$CW)
crabs$lnBD <- log(crabs$BD)

# train the WW SVM, with its default settings, on the crabs data

WWsvm <- trainSVM(crabs[,10:14],crabs$grp)

# Get in-sample classification results

WWpred <- predict(WWsvm,newdata=crabs[,10:14])
WWpred

# Compare classifications with true assignments

cat("Original classes:\n")
print(crabs$grp)
print(WWpred==crabs$grp)
```

predict.PVM

S3 predict method for class 'PVM'

Description

predict class probabilities based on objects of class 'PVM'.

Usage

```
## S3 method for class 'PVM'
predict(object, ..., newdata, eta="adjtogrid", probepsilon="adjtogrid",
trndt=NULL, retallprd=FALSE, newdataaskmat=FALSE)
```

Arguments

<code>object</code>	Object of class inheriting from “PVM”.
<code>...</code>	further arguments passed to or from other methods.
<code>newdata</code>	either a matrix or data frame with the data (with observations in rows and variables in columns) for which class probabilities are to be computed, or a Kernel matrix with the values resulting from applying the kernel function to all pairs between <code>newdata</code> and training data observations. In the later case argument <code>newdataasKmat</code> should be set to <code>TRUE</code> , and the <code>newdata</code> rows are associated with the new observations, while its columns correspond to training data observations.
<code>eta</code>	value of the eta hyper-parameter used in the conversion of SVM(s) predictions to class probabilities. It gives the relative importance of undesirable versus desirable l_1 norm deviations in the goodness of fit criterion optimized to find these probabilities. When set to the string “adjtograd” (default) it ensures that the undesirable deviations dominate the desirable ones, which is necessary for the consistency properties of the resulting estimators. See reference [1] for further details.
<code>probepsilon</code>	the minimum possible value for the class probabilities estimated by <code>predict.PVM</code> . It could be either the string “adjtograd” (default), a small strictly positive constant, or zero. When equal to “adjtograd” it is set to half the minimum weight provided by the grid matrix of weight specifications. This choice is consistent with the resolution implied by this matrix. If set to zero, it admits some class probabilities to be estimated exactly by zero. This choice implies that the corresponding classes will be impossible for the corresponding attribute vector values. See reference [1] for further discussion.
<code>trndt</code>	either a <code>NULL</code> value, a matrix or data frame with the training data (with observations in rows and variables in columns) or the string “newdata”, if (and only if) class probabilities are to be computed for the same data as the one used for training. When equal to <code>NULL</code> the training data will be retrieved from the value of argument <code>object</code> (if argument <code>newdataasKmat</code> is set to <code>FALSE</code>), or assumed to be already used in the computation of the kernel matrix given by argument <code>newdata</code> (if argument <code>newdataasKmat</code> is set to <code>TRUE</code>)
<code>retallprd</code>	a boolean flag stating if all weighted SVM predictions should be returned together with the estimated class probabilities.
<code>newdataasKmat</code>	a boolean flag stating if the new data is given as a pre-computed Kernel matrix.

Value

when `retallprd` is set to `FALSE`, a `ClassProb` object with the probability estimates for the new data. Class `ClassProb` are matrices or data frames of class probabilities (observations in rows, classes in columns), plus a class attribute, and have a `plot` method for relevant displays of these probabilities. When `retallprd` is set to `TRUE` a list with two components named (i) `Pprob`: a `ClassProb` object; and (ii) `classpred`: a matrix with observations in rows and class-weight specifications in columns, containing the weighted SVM predictions.

References

[1] Duarte Silva, A.P. (2025) Probabilistic Vector Machines. *Computers and Operations Research*, Vol. 183, 107203. <doi:10.1016/j.cor.2025.107203>

See Also

[trainPVM](#), [plot.ClassProb](#)

Examples

```
set.seed(1) # Fix the seed in order to get reproducible results
            # (given the the building ot the grid matrix has a random component)

## The following examples are based on the MASS data set "crabs".
## This data records physical measurements on 200 specimens of
## Leptograpsus variegatus crabs observed on the shores
## of Western Australia. The crabs are classified by two factors, sex and sp
## (crab species as defined by its colour: blue or orange), with two levels
## each. The measurement variables include the natural logarithms of carapace length (CL),
## the carapace width (CW), the size of the frontal lobe (FL) and the size of
## the rear width (RW). In the analysis provided, we created four classes
## by crossing the sex and sp levels.

library(MASS)
data(crabs)

crabs$grp <- interaction(crabs$sex,crabs$sp)

crabs$lnFL <- log(crabs$FL)
crabs$lnRW <- log(crabs$RW)
crabs$lnCL <- log(crabs$CL)
crabs$lnCW <- log(crabs$CW)
crabs$lnBD <- log(crabs$BD)

WWpvm <- trainPVM(crabs[,10:14],crabs$grp)
WWprob <- predict(WWpvm,newdata=crabs[,10:14],trndt="newdata")

# Display the probabilities of the predicted classes

plot(WWprob, , trdata=crabs[,10:14], grouping=crabs$grp)
plot(WWprob, , trdata=crabs[,10:14], grouping=crabs$grp, type="stckdbarplt")
WWprob

# Repeat the analysis, using the first 45 observations in each class for training,
# and the last 5 for probability estimation

trdata <- crabs[c(1:45,51:95,101:145,151:195),10:14]
trgrp <- crabs$grp[c(1:45,51:95,101:145,151:195)]
evaldata <- crabs[c(46:50,96:100,146:150,196:200),10:14]
WWpvm1 <- trainPVM(trdata,trgrp)
```

```
WWprob1 <- predict(WWpvm1,newdata=evaldata)

plot(WWprob1, type="stckdbarplt")
WWprob1
```

SetUpOptPar	<i>Set up optimization parameters.</i>
-------------	--

Description

SetUpOptPar sets up several control parameters for the optimization of the mathematical programming models used for SVM training.

Usage

```
SetUpOptPar(start=c("allub","alllb","compubwithlb","warmstarts"), alpha0=NULL,
             epsilon=1e-12, maxiter=100000, tol=1.5e-12 )
```

Arguments

start	string describing the initial solution for the optimization algorithm. It can have one of the following values: • “allub”: all variables are initialized at their upper bounds. • “alllb”: all variables are initialized at their lower bounds. • “compubwithlb”: the solutions with all variables at their lower bounds and with all variables at their upper bounds are compared, and the one with the best (lower) criterion value is selected as the initial solution. • warmstarts the initial solution is set at the values given by argument alpha0
alpha0	vector with initial values for the optimization algorithm. Only used wehn argument start is equal to “warmstarts”.
epsilon	stopping criterion for the optimization algorithm. When gradient values, or criteria increments, are below epsilon the algorithm stops, and the current solution is returned as the presumed optimum.
maxiter	maximum number of iterations for the optimization algorithm.
tol	variable numerical tolerance. Variables closer to their lower or upper bounds than tol are assumed to be exactly at the corresponding bounds.

Value

a list with components start, alpha0, epsilon, maxiter and tol .

See Also

[trainSVM](#)

 SetUpTunPar

Setting up SVM tuning parameters.

Description

SetUpTunPar sets up several control parameters in the procedure used for tuning SVM regularization hyper-parameters.

Usage

```
SetUpTunPar(tunex=NULL, tuneY=NULL, tuneK=NULL,
  Csrchpar=list(Cpowerbase=2.,Cgridinlev=7,Cnloops=2), crossval=FALSE,
  crossvalpar=list(Strfolds=TRUE,kfold=10,CVrep=3))
```

Arguments

tunex	a matrix or data frame containing the data (observations in rows and variables in columns) used for tuning SVM hyper-parameters. It is ignored if argument tuneK is not equal to NULL.
tuneY	a factor describing the response vector (with one label by observation) used for tuning SVM hyper-parameters.
tuneK	a kernel matrix based on the training data (columns), and the data used for tuning SVM hyper-parameters (rows). When not NULL, overrides the value of argument tunex.
Csrchpar	list of control parameters used for the search of the tuning parameters. It is formed by the following components: • Cpowerbase: base of successive powers defining the finite set of values to be searched. • Cgridinlev: number of powers of Cpowerbase in each loop. • Cnloops: number of times a search over Cgridinlev points is performed. Each additional loop moves from wider searches to more compact searches.
crossval	a logical flag indicating if the tuning procedure should be based on a (statistically more sound, but more time intensive) cross-validation strategy.
crossvalpar	list of control parameters used for the cross-validation strategy. It is ignored when crossval is set to FALSE (default), and is formed by the following components: • Strfolds: a logical flag indicating if the folds should be stratified according to the original class proportions (default), or randomly generated from the whole training sample, ignoring class membership. • kfold: the number of training sample folds to be created in each replication. • CVrep: the number of replications to be performed.

Details

SetUpTunPar sets up several control parameters in the procedure used for tuning SVM regularization hyper-parameters.

The tuning procedure is based on a search for the optimal value of an performance criterion over a finite grid of `Csrchpar$Cgridinlev` different powers of `Csrchpar$Cpowerbase`, centered at zero.

When argument `Csrchpar$Cnloops` is set to one, these powers are consecutive integers. When `Csrchpar$Cnloops` is higher than one, there is an initial search over a wider grid, which is then re-fined around the best value found in the previous loop. This procedure is repeated `Csrchpar$Cnloops` times, in such way that in the last loop there are `Csrchpar$Cgridinlev` consecutive integers.

The evaluation criterion to be optimized can be an SVM error rate on its training data (if arguments `tunex` and `tuneK` are both NULL), on a tuning data set defined by arguments `tunex` and `tuneK` or `tuneK` (if argument `crossval` is FALSE), or a cross-validated estimate of the SVM error rate (if argument `crossval` is TRUE) as defined by argument `crossvalpar`.

Value

a list with components `tunex`, `tuneK`, `Csrchpar`, `crossval` and `crossvalpar`.

See Also

[trainSVM](#)

trainPVM

Training of Probabilistic Vector Machines

Description

trainPVM creates objects of class 'PVM' (Probabilistic Vector Machines) by training a sequence of kernel-based weighted Supported Vector Machines with different class-weight specifications.

Usage

```
trainPVM(x=NULL, y, scaled=TRUE, K=NULL, loss=c("WW","LLW"), withbias=FALSE,
         kernel=c("rbfdot","vanilladot","polydot"), kpar=list(sigma="d2median"),
         C=0.25, lambda=NULL, tunex=x, tuneK=K,
         grid=NULL, dpiinv=ceiling(sqrt(length(y))/0.2), keepdt=TRUE, ... )
```

Arguments

<code>x</code>	a matrix or data frame containing the training data, with observations in rows and variables in columns. If <code>x</code> is equal to NULL, a kernel matrix should be provided directly in argument <code>K</code> .
<code>y</code>	a factor describing the response vector, with one label by observation.
<code>scaled</code>	a logical flag indicating wheter or not the variables should be scaled to unit variance.

K	a symmetrix kernel matrix. When equal to NULL, the training data should be given by argument x and the kernel matrix is computed from it, and from the values of arguments kernel and kpar. When K is not NULL, it is assumed to be equal to a pre-computed kernel matrix and the arguments x, kernel, and kpar are ignored.
loss	a string specifying the multiclass large margin loss to be employed. Currently the following alternatives are implemented: “LLW” for the Lee, Lin and Wahba loss (see reference[3]), and “WW” for the Weston and Watkins loss (see reference[4]). In two-class problems these two losses are equivalent, and also equivalent to the classical hinge loss.
withbias	a logical flag indicating if a bias term (intercept) should be included in the classification rule. For problems with more than two classes, currently only rules without bias terms are implemented.
kernel	the kernel function used for training and prediction. Currently this argument can be set to one of the following strings: • rbfdot Radial Basis kernel “Gaussian” • vanilladot Linear kernel • polydot Polynomial kernel
kpar	the list of kernel hyper-parameters. This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • sigma: inverse kernel width for the Radial Basis kernel function “rbf-dot”. Either a numeric value or one of the strings “d2median” (default), “d2q01q09mean”, or “d2q01q09hmean”. In the three later cases this value is computed automatically, from the distribution of all euclidean distances between dt1 observation pairs. See GetrbfdotSigPar for details. • degree, scale, offset: for the Polynomial kernel “polydot”
C	cost of constraints violation in the base SVMs (default: 0.25). This is the “C”-constant of the regularization term in the Lagrange formulation. When set to the string “tuneit” the value of “C” is tuned to the data provided in arguments tuneK (default), or tunex and tuneY (if tuneK is set to NULL). The value of this argument is overridden if the argument lambda has a value different than NULL.
lambda	regularization parameter in the functional analysis formulation of SVMs. When set to a non-NULL value, overrides the value of argument C. Specifying the amount of regularization by arguments C or lambda leads to equivalent models if the corresponding constants satisfy the relation $\lambda = 1/(2 n C)$, where n is the training sample size. When set to the string “tuneit” the value of lambda is tuned to the data provided in arguments tuneK (default), or tunex and tuneY (if tuneK is set to NULL).
tunex	a matrix or data frame containing the data (observations in rows and variables in columns) used to tune the hyper-parameters C and lambda. It is ignored if none of these arguments is set to the string “tuneit”, or if argument tuneK is not equal to NULL.
tuneY	a factor describing the response vector (with one label by observation) used to tune the hyper-parameters C and lambda. It is ignored if none of these arguments is set to the string “tuneit”.

tuneK	a kernel matrix based on the training data (columns), and the data used (rows) for tuning the hyper-parameters C and λ . It is ignored if none of these arguments is set to the string “tuneit”. When set to a non-NULL value overrides the value of argument <code>tunex</code> .
grid	A matrix with the class-weight specifications used to train the PVM. The rows correspond to different specifications, and the columns to the corresponding class weights. All rows should have non-negative elements adding up to one. When set to NULL (default) the grid matrix is automatically created based on the value of argument <code>dpiinv</code> .
dpiinv	the distance between two consecutive weights in the (one-dimensional) specifications of class-weights used to construct a grid matrix. Note that weights are uniformly distributed only along one dimension, while the remaining components of the grid are set at random. However, all grid dimensions (classes) are chosen in turn as the one forced to have uniform weights. See reference [2] for further details. This argument is ignored when argument <code>grid</code> is not NULL.
keepdt	a logical flag indicating if the original training data should be returned together with the trained PVM
...	other arguments to be passed to <code>trainSVM</code>

Details

`trainPVM` trains the Probabilistic Vector Machines (PVMs) proposed by Duarte Silva in reference [2]. PVMs are distribution-free estimators of class probabilities based on the predictions given by a sequence of kernel-based weighted Supported Vector Machines (SVMs) with different class-weight specifications. A grid matrix with these specifications is usually created automatically, with its resolution level controlled by the argument `dpiinv`, but can also be directly provided by argument `grid`.

Currently there are two variants of multiclass PVMs implemented in `trainPVM`: (i) machines that use the SVM loss proposed by Lee, Lin and Wahba (see reference [3]), and machine that use the loss proposed by Weston and Watkins (see reference [4]). The former variant has better asymptotic properties, but there are some empirical evidence suggesting that the latter often gives more reliable results in many applications (see, e.g., reference [1]). For two-class problems these two variants are equivalent.

In problems with more than two classes currently `trainPVM` only implements PVMs based on classification rules derived from weighted kernel-based SVMs without bias terms. In two-class problems rules with or without (default) bias terms can be used, with this choice controlled by the argument `with bias`. It can be shown that in kernel-based SVMs dropping the bias terms has some computational advantages, and it has been argued that it should not have a noticeable impact on statistical performance (see references [1] and [2] for further discussion).

The amount of regularization provided by the underlying SVMs is controlled by the value of the hyper-parameters C or λ . While C (the cost of constraint violations) is the traditional regularization hyper-parameter used in the majority of the SVM literature, there are also alternative formulations in which SVM training is presented as a particular case of regularized function estimation in functional analysis. In this perspective, the SVM constraint violations are viewed as unit-cost losses, and the margin component of the traditional SVM criterion is understood as a complexity penalty that is weighted by the regularization hyper-parameter λ . The two formulations are mathematically equivalent if λ is set to $1/(2 n C)$, or if C is set to $1/(2 n \lambda)$, with n being

the number of observations in the training data. Note that, as C and λ are inversely related, higher values of λ and lower values of C correspond to higher regularization and smoother classification rules, while lower λ values and higher C values lead to more complex (and flexible) rules with better training sample performance, but possibly with worse generability potential.

In trainPVM the values of C and λ can be set by the user, or tuned automatically to the training or additional data, provided by arguments `tunex`, `tuney`, or `tuneK`. In that case, C and λ are found by optimizing the log-likelihood of the class probability estimates on the tuning data. However, the tuning procedure is computationally intensive, and can be very time consuming. See reference [2] for further details.

Value

A object of class “PVM” containing the trained Probability Vector Machine. Class “PVM” has a `predict` method, that estimates class probabilities from “PVM” objects, and data frames or Kernel matrices of new (or training) data.

An object inheriting from class “PVM” is a list containing at least the following components:

nclasses the number of different classes.

kernel the value of argument `kernel` in the call to `trainPVM`.

kpar the value of argument `kpar` in the call to `trainPVM`.

grid the grid matrix with the class-weight specifications used to train the PVM.

svms a list with as many components as the number of different weighted SVMs trained. Each component of this list is an object of class `kernelSVM` (if argument `withbias` is `FALSE`) or class `kernelksvms` (if argument `withbias` is `TRUE`) containing the trained SVM for a particular weight specification.

References

- [1] Dogan U.; Glasmachers T. and Igel, C. (2016) A unified view on multi-class support vector classification. *Journal of Machine Learning Research*, Vol. 17 (45), 1–32.
- [2] Duarte Silva, A.P. (2025) Probabilistic Vector Machines. *Computers and Operations Research*, Vol. 183, 107203. <doi:10.1016/j.cor.2025.107203>
- [3] Lee, Y.; Lin, Y. and Wahba, G. (2004) Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, Vol. 99, 67–81. <doi:10.1198/016214504000000098>
- [4] Weston, J. and Watkins, C. (1999) Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 219–224.

See Also

[predict.PVM](#), [trainSVM](#), [SetUpOptPar](#), [GetrbfdotSigPar](#), [plot.ClassProb](#)

Examples

```
set.seed(1) # Fix the seed in order to get reproducible results
            # (given the the building of the grid matrix has a random component)
```

```

## The following examples are based on the MASS data set "crabs".
# This data records physical measurements on 200 specimens of
# Leptograpsus variegatus crabs observed on the shores
## of Western Australia. The crabs are classified by two factors, sex and sp
## (crab species as defined by its colour: blue or orange), with two levels
## each. The measurement variables include the natural logarithms of carapace length (CL),
## the carapace width (CW), the size of the frontal lobe (FL) and the size of
## the rear width (RW). In the analysis provided, we created four classes
# by crossing the sex and sp levels.

library(MASS)
data(crabs)

crabs$grp <- interaction(crabs$sex,crabs$sp)

crabs$lnFL <- log(crabs$FL)
crabs$lnRW <- log(crabs$RW)
crabs$lnCL <- log(crabs$CL)
crabs$lnCW <- log(crabs$CW)
crabs$lnBD <- log(crabs$BD)

# Estimate class probabilities based on the WW loss

WWpvm <- trainPVM(crabs[,10:14],crabs$grp)
WWprob <- predict(WWpvm,newdata=crabs[,10:14],trndt="newdata")

# Display the probabilities of the predicted classes

plot(WWprob, , trdata=crabs[,10:14], grouping=crabs$grp)
plot(WWprob, , trdata=crabs[,10:14], grouping=crabs$grp, type="stckdbarplt")
WWprob

# Repeat the analysis, using the first 45 observations in each class for training,
# and the last 5 for probability estimation

trind <- c(1:45,51:95,101:145,151:195)
evalind <- c(46:50,96:100,146:150,196:200)
trdata <- crabs[trind,10:14]
trgrp <- crabs$grp[trind]
evaldata <- crabs[evalind,10:14]
WWpvm1 <- trainPVM(trdata,trgrp)
WWprob1 <- predict(WWpvm1,newdata=evaldata)

plot(WWprob1, type="stckdbarplt")
WWprob1

## Repeat the analysis by using a pre-computed kernel matrix given by the K argument
## (Note: this is recommended in multiple analysis based on the same data, in order
## to avoid unnecessary repeated evaluations of the kernel function

```

```

set.seed(1) # To ensure that the results agree with the previous analysis
            # (i.e. forcing the same grid matrix)

# All data as training data

GramMat <- makeKMat(scale(crabs[,10:14],center=FALSE),kernel="rbfdot",kpar=list(sigma="d2median"))
# Gram matrix, with sigma automatically adjusted to the scaled training data

WWpvm2 <- trainPVM(K=GramMat,y=crabs$grp,)
WWprob2 <- predict(WWpvm2,newdata=GramMat,newdataaskmat=TRUE)
plot(WWprob2, type="stckdbarplt")

# Split 45 observations per group for training and 5 observations per group for prediction

scltrdata <- scale(crabs[trind,10:14],center=FALSE)
sclevaldata <- scale(crabs[evalind,10:14],center=FALSE,scale=attr(scltrdata,"scaled:scale"))

GramMat <- makeKMat(scltrdata,kernel="rbfdot",kpar=list(sigma="d2median"))
# Gram matrix, with sigma automatically adjusted to the scaled training data
trnevalKernMat <- makeKMat(scltrdata,sclevaldata,kernel="rbfdot",kpar=list(sigma="d2median"))
#Kernel matrix between the (commonly scaled) training and evaluation data

WWpvm3 <- trainPVM(K=GramMat,y=crabs$grp[trind])
WWprob3 <- predict(WWpvm3,newdata=trnevalKernMat,newdataaskmat=TRUE)
plot(WWprob3, type="stckdbarplt")

```

trainSVM

Efficient training of (multiclass) kernel-based Support Vector Machines

Description

trainSVM creates objects of class kernelSVM or class kernelSVMs by training one or several kernel-based (possibly weighted) Support Vector Machines for general multiclass problems, using an 'one-in-all' global model approach.

Usage

```

trainSVM(x=NULL, y, class.weights=rep(1.,length(levels(y))), scaled=TRUE, K=NULL,
         loss=c("WW","LLW"), kernel=c("rbfdot","vanilladot","polydot"),
         kpar=list(sigma="d2median"), C=1., lambda=NULL, keepdt=TRUE, retotpst=FALSE,
         OptCntrl=SetUpOptPar(), TunCntrl=SetUpTunPar() )

```

Arguments

<code>x</code>	a matrix or data frame containing the training data, with observations in rows and variables in columns. If <code>x</code> is equal to <code>NULL</code> , a kernel matrix should be provided directly in argument <code>K</code> .
<code>y</code>	a factor describing the response vector, with one label by observation.
<code>class.weights</code>	a vector or a matrix of class weights. When <code>class.weights</code> is a vector, its length should equal to number of classes, and the <i>i</i> -th component represents the weight given to the <i>i</i> -th class in the SVM loss function. For classical (non-weighted) SVMs all its components are equal to one (default). In weighted SVM variants, the <code>class.weights</code> components may be different and should add up to one. When <code>class.weights</code> is a matrix the number of columns should equal the number of classes, and each row corresponds to a different weight specification. In that case the resulting SVM is trained as many times as the number of rows. When <code>class.weights</code> is a vector or a single row matrix, <code>trainSVM</code> returns an object of class <code>kernelSVM</code> (one trained SVM), and when <code>class.weights</code> is a matrix with multiple rows, <code>trainSVM</code> returns an object of class <code>kernelSVMs</code> (several trained SVMs).
<code>scaled</code>	A logical flag indicating whether or not the variables should be scaled to unit variance.
<code>K</code>	A symmetric kernel matrix. When equal to <code>NULL</code> , the training data should be given by argument <code>x</code> and the kernel matrix is computed from it and from the values of arguments <code>kernel</code> , and <code>kpar</code> . When <code>K</code> is not <code>NULL</code> , it is assumed to be equal to a pre-computed kernel matrix and the arguments <code>x</code> , <code>kernel</code> , and <code>kpar</code> are ignored.
<code>loss</code>	a string specifying the multiclass large margin loss to be employed. Currently the following alternatives are implemented: “WW” for the Weston and Watkins loss (see reference [6]), and “LLW” for the Lee, Lin and Wahba loss (see reference [4]). In two-class problems these two losses are equivalent and also equivalent to the classical hinge loss. In problems with more than two classes these losses differ and the “LLW” loss has better asymptotic properties, but the “WW” loss was found empirically to usually give more reliable results in many practical applications.
<code>kernel</code>	the kernel function used for training and prediction. Currently this argument can be set to one of the following strings: • “rbfdot” Radial Basis kernel “Gaussian” • “vanilladot” Linear kernel • “polydot” Polynomial kernel
<code>kpar</code>	the list of kernel hyper-parameters. This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • <code>sigma</code>: inverse kernel width for the Radial Basis kernel function “rbfdot”. Either a numeric value or one of the strings “d2median” (default), “d2q01q09mean”, or “d2q01q09hmean”. In the three latter cases this value is computed automatically, from the distribution of all euclidean distances between <code>dt1</code> observation pairs. See GetrbfdotSigPar for details. • <code>degree</code>, <code>scale</code>, <code>offset</code>: for the Polynomial kernel “polydot”

C	cost of constraints violation (default: 1). This is the "C"-constant of the regularization term in the Lagrange formulation. When set to the string "tuneit" the value of "C" is tuned according to the procedure defined by the value of argument TunCntrl. The value of this argument is overridden if the argument lambda has a valued different than NULL.
lambda	regularization parameter in the functional analysis formulation of SVMs. When set to a non-NULL value, overrides the value of argument C. Specifying the amount of regularization by arguments C or lambda leads to equivalent models if the corresponding constants satisfy the relation $\lambda = 1/(2 n C)$, where n is the training sample size. When set to the string "tuneit" the value of lambda is tuned according to the procedure defined by the value of argument TunCntrl.
keepdt	a logical flag indicating if the original training data should be returned together with the trained SVM(s)
retotpst	a logical flag indicating if internal optimization results and statistics should be returned together with the trained SVM(s)
OptCntrl	a list with several control arguments for the optimization algorithm used in the SVM(s) training. See SetUpOptPar for details.
TunCntrl	a list with several arguments controlling the hyperparameter (C or lambda) tuning. Only used when either C or lambda are set to the string "tuneit". See SetUpTunPar for details.

Details

trainSVM trains multiclass kernel-based (possibly weighted) Supported Vector Machines (SVMs) without bias terms. It implements Duarte Silva's adaptation (see reference [3]) of Dogan, Glasmachers and Igel's efficient training algorithm (see reference [1]) for 'all-in-one' multiclass SVMs. It can be shown that in kernel-based SVMs dropping bias terms has considerable computational advantages, and it has been argued that the resulting rules have essentially the same statistical properties (at least asymptotically for universal kernels) as the corresponding rules with bias terms. See references [2], [3] and [5] for further discussion.

Currently there are two variants of multiclass SVMs implemented in trainSVM: (i) machines that use the SVM loss proposed by Weston and Watkins (see reference [6]), and machines that use the loss proposed by Lee, Lin and Wahba (see reference [4]). The later variant has better asymptotic properties, but there are some empirical evidence suggesting that the former often gives more reliable results in many applications (see, e.g. reference [2]). For two-class problems these two variants are equivalent among themselves and to the classical SVM hinge loss.

The amount of regularization of the resulting SVMs is controled by the value of the hyper-parameters C or lambda. While C (the cost of constraint violations) is the traditional regularization hyper-parameter used in the majority of the SVM literature, there are also alternative formulations in which SVM training is presented as a particular case of regularized function estimation in functional analysis. Under this perspective, the SVM constraint violations are viewed as unit-cost losses, and the margin component of the traditional SVM criterion is understood as a complexity penalty that is weighted by the regularization hyper-parameter lambda. The two formulations are mathematically equivalent if lambda is set to $1/(2 n C)$, or if C is set to $1/(2 n \lambda)$, with n being the number of observations in the training data. Note that, as C and lambda are inversely related, higher values of lambda and lower values of C correspond to higher regularization and smoother classification

rules, while lower lambda values and higher C values lead to more complex (and flexible) rules, with better training sample performance, but potentially worse generability ability.

In `trainSVM` the values of `C` and `lambda` can be set by the user, or tuned automatically to the training or tuning data according to the procedure defined by the value of argument `TunCntrl`. See the documentation of [SetUpTunPar](#) for further details.

Value

An object inheriting from class “kernelSVM” containing the trained Support Vector Machine(s). It could be either an “kernelSVM” object if argument `class.weights` is a vector or a single-row matrix, or an “kernelSVMs” object otherwise. Class “kernelSVM” has a `predict` method, that gives class predictions from “kernelSVM” objects, and data frames or kernel matrices of new (or training) data.

An object inheriting from class “kernelSVM” is a list containing, at least, the following components:

alpha in “kernelSVM” classes, a matrix (observations in rows, classes in columns) containing the resulting support vectors. In “kernelSVMs” classes, a three-dimensional array, in which each level of the third dimension corresponds to a different trained SVM, and the first two dimensions contain the corresponding support vectors, with observations in rows and classes in columns.

grplvls the levels of the factor representing the different classes.

lambda the value of the lambda regularization hyperparameter used in the SVM(s) training.

C the value of the C regularization hyperparameter used in the SVM(s) training.

x when argument `keepdt` is `TRUE`, a matrix or data frame containing the training data, with observations in rows and variables in columns. Otherwise, the value `NULL`.

scale when argument `scaled` is `TRUE`, a vector with the variables standard deviations, that were used for data scaling. Otherwise, the value `NULL`.

kernel the value of argument `kernel` in the call to `trainSVM`.

kpar the value of argument `kpar` in the call to `trainSVM`.

optlist this component is only non `NULL` when argument `retotpst` is set to `TRUE`. In that case it is a list with the following components:

optvalue the returned value of the optimization model criterion used for SVM training .

iterations the number of iterations used by the SVM training optimization algorithm.

hitrate the classification hit rate (when known) of the trained SVM. Typically this value is only available when arguments `C` or `lambda` are set to the string “tuneit”, and refers to the optimal hit rate obtained in the tuning data. When `C` or `lambda` is not tuned, this component is set to `NULL`.

References

- [1] Dogan, U.; Glasmachers, T. and Igel, C. (2011) Fast training of multi-class support vector machines. *Technical report. Department of Computer Sciences, University of Copenhagen*.
- [2] Dogan U.; Glasmachers T. and Igel, C. (2016) A unified view on multi-class support vector classification. *Journal of Machine Learning Research*, Vol. 17 (45), 1–32.

- [3] Duarte Silva, A.P. (2025) Probabilistic Vector Machines. *Computers and Operations Research*, Vol. 183, 107203. <doi:10.1016/j.cor.2025.107203>
- [4] Lee, Y.; Lin, Y. and Wahba, G. (2004) Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, Vol. 99, 67–81. >doi:10.1198/016214504000000098>
- [5] Poggio, T.; Mukherjee, S.; Rifkin, R.; Raklin, A. and Verri, A. (2002). B. In *Uncertainty in Geometric Computations*, Springer US.
- [6] Weston, J. and Watkins, C. (1999) Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 219–224.

See Also

[predict.kernelSVM](#), [SetUpOptPar](#), [SetUpTunPar](#), [GetrbfdotSigPar](#).

Examples

```
## The following examples are based on the MASS data set "crabs".
# This data records physical measurements on 200 specimens of
# Leptograpsus variegatus crabs observed on the shores
## of Western Australia. The crabs are classified by two factors, sex and sp
## (crab species as defined by its colour: blue or orange), with two levels
## each. The measurement variables include the natural logarithms of carapace length (CL),
## the carapace width (CW), the size of the frontal lobe (FL) and the size of
## the rear width (RW). In the analysis provided, we created four classes
# by crossing the sex and sp levels.

library(MASS)
data(crabs)

crabs$grp <- interaction(crabs$sex,crabs$sp)

crabs$lnFL <- log(crabs$FL)
crabs$lnRW <- log(crabs$RW)
crabs$lnCL <- log(crabs$CL)
crabs$lnCW <- log(crabs$CW)
crabs$lnBD <- log(crabs$BD)

# train the WW and LLW SVMs, with their default settings, on the crabs data

WWsvm <- trainSVM(crabs[,10:14],crabs$grp,loss="WW")
LLWsvm <- trainSVM(crabs[,10:14],crabs$grp,loss="LLW")

# Get in-sample classification results

WWpred <- predict(WWsvm,newdata=crabs[,10:14])
WWpred
LLWpred <- predict(LLWsvm,newdata=crabs[,10:14])
LLWpred
```

```

# Compare classifications with true assignments

print(WWpred==crabs$grp)
print(LLWpred==crabs$grp)

# Repeat the analysis, by tuning
# the regularization hyper-parameter to the training data

WWsvm1 <- trainSVM(crabs[,10:14],crabs$grp,C="tuneit")
WWpred1 <- predict(WWsvm1,newdata=crabs[,10:14])
WWpred1
print(WWpred1==crabs$grp)

LLWsvm1 <- trainSVM(crabs[,10:14],crabs$grp,C="tuneit",loss="LLW")
LLWpred1 <- predict(LLWsvm1,newdata=crabs[,10:14])
LLWpred1
print(LLWpred1==crabs$grp)

# Repeat the analysis, for the WW loss only, using the first 45 observations
# in each class for training, and the last 5 for prediction

trind <- c(1:45,51:95,101:145,151:195)
evalind <- c(46:50,96:100,146:150,196:200)
trdata <- crabs[trind,10:14]
trgrp <- crabs$grp[trind]
evaldata <- crabs[evalind,10:14]
WWsvm2 <- trainSVM(trdata,trgrp,C="tuneit")
WWpred2 <- predict(WWsvm2,newdata=evaldata)
WWpred2
print(WWpred2==crabs$grp[evalind])

# Now, tune C by a cross-validated estimate of the error rate in the training data

WWsvm3 <- trainSVM(trdata,trgrp,C="tuneit",TunCntrl=list(crossval=TRUE))
WWpred3 <- predict(WWsvm3,newdata=evaldata)
WWpred3
print(WWpred3==crabs$grp[evalind])

## Repeat the analysis by using a pre-computed kernel matrix given by the K argument
## (Note: this is recommended in multiple analysis based on the same data, in order
## to avoid unnecessary repeated evaluations of the kernel function)

# All data as training data

GramMat <- makeKMat(scale(crabs[,10:14],center=FALSE),kernel="rbfdot",kpar=list(sigma="d2median"))
# Gram matrix, with sigma automatically adjusted to the scaled training data

WWsvm4 <- trainSVM(K=GramMat,y=crabs$grp,C="tuneit")

```

```
WWpred4 <- predict(WWsvm4,KMat=GramMat)
print(WWpred4==crabs$grp)

# Split 45 observations per group for training and 5 observations per group for prediction

scltrdata <- scale(crabs[trind,10:14],center=FALSE)
sclevaldata <- scale(crabs[evalind,10:14],center=FALSE,scale=attr(scltrdata,"scaled:scale"))

GramMat <- makeKMat(scltrdata,kernel="rbfdot",kpar=list(sigma="d2median"))
# Gram matrix, with sigma automatically adjusted to the scaled training data
trnevalKernMat <- makeKMat(scltrdata,sclevaldata,kernel="rbfdot",kpar=list(sigma="d2median"))
#Kernel matrix between the (commonly scaled) training and evaluation data

WWsvm5 <- trainSVM(K=GramMat,y=crabs$grp[trind],C="tuneit")
WWpred5 <- predict(WWsvm5,KMat=trnevalKernMat)
print(WWpred5==crabs$grp[evalind])
```

Index

* **ProbSVMs**

- ProbSVMs-package, [2](#)
- as.ClassProb, [4](#), [9](#)
- buildgrid (ProbSVMs-package), [2](#)
- Csearch (ProbSVMs-package), [2](#)
- evalPP (ProbSVMs-package), [2](#)
- GausKern (ProbSVMs-package), [2](#)
- GenCrossVFolds (ProbSVMs-package), [2](#)
- GetrbfdotSigPar, [5](#), [6](#), [17](#), [19](#), [22](#), [25](#)
- is.wholenumber (ProbSVMs-package), [2](#)
- l2norm2 (ProbSVMs-package), [2](#)
- LPdual (ProbSVMs-package), [2](#)
- makeKMat, [6](#)
- makeKMat0 (ProbSVMs-package), [2](#)
- MedianDist1Set (ProbSVMs-package), [2](#)
- MedianDist2Sets (ProbSVMs-package), [2](#)
- plot.ClassProb, [3](#), [4](#), [7](#), [13](#), [19](#)
- predict.kernelSVM, [3](#), [10](#), [25](#)
- predict.PVM, [3](#), [9](#), [11](#), [19](#)
- ProbSVMs (ProbSVMs-package), [2](#)
- ProbSVMs-package, [2](#)
- SetUpOptPar, [14](#), [19](#), [23](#), [25](#)
- SetUpTunPar, [15](#), [23–25](#)
- sigest1 (ProbSVMs-package), [2](#)
- trainandevalPVM (ProbSVMs-package), [2](#)
- trainPVM, [2](#), [3](#), [6](#), [9](#), [13](#), [16](#)
- trainSVM, [2](#), [3](#), [6](#), [10](#), [14](#), [16](#), [19](#), [21](#)