Package 'WeightSVM'

October 12, 2024

Version 1.7-16

Title Subject Weighted Support Vector Machines

Imports graphics, grDevices, stats, methods, utils

Suggests SparseM, xtable, Matrix, MASS, e1071, knitr, slam, kernlab

Maintainer Tianchen Xu <tx2155@columbia.edu>

Description Functions for subject/instance weighted support vector machines (SVM).
It uses a modified version of 'libsvm' and is compatible with package 'e1071'. It also allows user defined kernel matrix.

URL https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#weights_for_data_instances

BugReports https://github.com/zjph602xtc/wsvm/issues

License GPL-2 | GPL-3

LazyLoad yes

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

Date/Publication 2024-10-12 03:30:02 UTC

Author Tianchen Xu [aut, cre] (<https://orcid.org/0000-0002-0102-7630>),
Chih-Chung Chang [ctb, cph] (libsvm C++-code),
Chih-Chen Lin [ctb, cph] (libsvm C++-code),
Ming-Wei Chang [ctb, cph] (libsvm C++-code),
Hsuan-Tien Lin [ctb, cph] (libsvm C++-code),
Ming-Hen Tsai [ctb, cph] (libsvm C++-code),
Chia-Hua Ho [ctb, cph] (libsvm C++-code),
Hsiang-Fu Yu [ctb, cph] (libsvm C++-code),
David Meyer [ctb],
Evgenia Dimitriadou [ctb],
Kurt Hornik [ctb],
Andreas Weingessel [ctb],
Friedrich Leisch [ctb]

Contents

`plot.tune_wsvm` *Plot Tuning Object*

Description

Visualizes the results of parameter tuning.

Usage

```
## S3 method for class 'tune_wsvm'
plot(x, type = c("contour", "perspective"), theta = 60,
          col = "lightblue", main = NULL, xlab = NULL, ylab = NULL,
          swapxy = FALSE, transform.x = NULL, transform.y = NULL,
          transform.z = NULL, color.palette = hsv_palette(),
          nlevels = 20, ...)
```

Arguments

`x`	an object of class `tune_wsvm`
`type`	choose whether a contour plot or a perspective plot is used if two parameters are to be visualized. Ignored if only one parameter has been tuned.
`theta`	angle of azimuthal direction.
`col`	the color(s) of the surface facets. Transparent colors are ignored.
`main`	main title.
`xlab, ylab`	titles for the axes. N.B. These must be character strings; expressions are not accepted. Numbers will be coerced to character strings.
`swapxy`	if `TRUE`, the parameter axes are swaped (only used in case of two parameters).
`transform.x, transform.y, transform.z`	functions to transform the parameters (`x` and `y`) and the error measures (`z`). Ignored if `NULL`.
`color.palette`	color palette used in contour plot.
`nlevels`	number of levels used in contour plot.
`...`	Further graphics parameters.

Value

None

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
Modified by Tianchen Xu <tx2155@columbia.edu>

See Also

tune_wsvm

Examples

```
data(iris)

obj <- tune_wsvm(Species~., weight = c(rep(0.8, 50),rep(1,100)),
            data = iris, ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
            tunecontrol = tune.control(sampling = "fix"))

summary(obj)
plot(obj, transform.x = log2, transform.y = log2)
plot(obj, type = "perspective", theta = 120, phi = 45)
```

---

plot.wsvm *Plot WSVM Objects*

Description

Generates a scatter plot of the input data of a wsvm fit for classification models by highlighting the classes and support vectors. Optionally, draws a filled contour plot of the class regions.

Usage

```
## S3 method for class 'wsvm'
plot(x, data, formula, fill = TRUE, grid = 50, slice = list(),
symbolPalette = palette(), svSymbol = "x", dataSymbol = "o", ...)
```

Arguments

x	An object of class wsvm
data	data to visualize. Should be the same used for fitting.
formula	formula selecting the visualized two dimensions. Only needed if more than two input variables are used.
fill	switch indicating whether a contour plot for the class regions should be added.
grid	granularity for the contour plot.

slice	a list of named values for the dimensions held constant (only needed if more than two variables are used). The defaults for unspecified dimensions are 0 (for numeric variables) and the first level (for factors). Factor levels can either be specified as factors or character vectors of length 1.
symbolPalette	Color palette used for the class the data points and support vectors belong to.
svSymbol	Symbol used for support vectors.
dataSymbol	Symbol used for data points (other than support vectors).
...	additional graphics parameters passed to filled.contour and plot.

Value

None

Author(s)

David Meyer
Modified by Tianchen Xu <tx2155@columbia.edu>

See Also

wsvm

Examples

```
## a simple example
data(cats, package = "MASS")
m <- wsvm(Sex~., data = cats, weight = rep(1,144))
plot(m, cats)

## more than two variables: fix 2 dimensions
data(iris)
m2 <- wsvm(Species~., data = iris, weight = rep(1,150))
plot(m2, iris, Petal.Width ~ Petal.Length,
     slice = list(Sepal.Width = 3, Sepal.Length = 4))

## plot with custom symbols and colors
plot(m, cats, svSymbol = 1, dataSymbol = 2, symbolPalette = rainbow(4),
color.palette = terrain.colors)
```

---

predict.wsvm *Predict Method for Subject Weighted Support Vector Machines*

Description

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

Usage

```
## S3 method for class 'wsvm'
predict(object, newdata, decision.values = FALSE,
probability = FALSE, ..., na.action = na.omit)
```

Arguments

object	Object of class "wsvm", created by wsvm.
newdata	An object containing the new input data: either a matrix or a sparse matrix (object of class Matrix provided by the **Matrix** package, or of class matrix.csr provided by the **SparseM** package, or of class simple_triplet_matrix provided by the **slam** package). A vector will be transformed to a n x 1 matrix.
decision.values	Logical controlling whether the decision values of all binary classifiers computed in multiclass classification shall be computed and returned.
probability	Logical indicating whether class probabilities should be computed and returned. Only possible if the model was fitted with the probability option enabled.
na.action	A function to specify the action to be taken if 'NA's are found. The default action is na.omit, which leads to rejection of cases with missing values on any required variable. An alternative is na.fail, which causes an error if NA cases are found. (NOTE: If given, this argument must be named.)
...	Currently not used.

Value

A vector of predicted values (for classification: a vector of labels, for density estimation: a logical vector). If decision.value is TRUE, the vector gets a "decision.values" attribute containing a n x c matrix (n number of predicted values, c number of classifiers) of all c binary classifiers' decision values. There are k * (k - 1) / 2 classifiers (k number of classes). The colnames of the matrix indicate the labels of the two classes. If probability is TRUE, the vector gets a "probabilities" attribute containing a n x k matrix (n number of predicted values, k number of classes) of the class probabilities.

Note

If the training set was scaled by wsvm (done by default), the new data is scaled accordingly using scale and center of the training data.

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
Modified by Tianchen Xu <tx2155@columbia.edu>

See Also

wsvm

Examples

```
## load dataset
data(iris)
attach(iris)

## classification mode
# default with factor response:
model1 <- wsvm(Species ~ ., weight = rep(1,150), data = iris) # same weights
model2 <- wsvm(x = iris[,1:4], y = iris[,5],
               weight = c(rep(0.08, 50),rep(1,100))) # less weights to setosa
x <- subset(iris, select = -Species)
y <- iris$Species
model3 <- wsvm(x, y, weight = rep(10,150)) # similar to model 1, but larger weights for all subjects

# test with train data
pred <- predict(model1, iris[,1:4])
# (same as:)
pred <- fitted(model1)

# Check accuracy:
table(pred, y) # model 1, equal weights

# compute decision values and probabilities:
pred <- predict(model1, x, decision.values = TRUE)
attr(pred, "decision.values")[1:4,]


## try regression mode on two dimensions
# create data
x <- seq(0.1, 5, by = 0.05)
y <- log(x) + rnorm(x, sd = 0.2)

# estimate model and predict input values
model1 <- wsvm(x, y, weight = rep(1,99))
model2 <- wsvm(x, y,
        weight = seq(99,1,length.out = 99)) # decreasing weights

# visualize
plot(x, y)
points(x, log(x), col = 2)
points(x, fitted(model1), col = 4)
points(x, fitted(model2), col = 3) # better fit for the first few points
```

`tune.control` *Control Parameters for the tune/tune_wsvm Function*

Description

Creates an object of class `tune.control` to be used with the `tune`/`tune_wsvm` function, containing various control parameters.

Usage

```
tune.control(random = FALSE, nrepeat = 1, repeat.aggregate = mean,
sampling = c("cross", "fix", "bootstrap"), sampling.aggregate = mean,
sampling.dispersion = sd,
cross = 10, fix = 2/3, nboot = 10, boot.size = 9/10, best.model = TRUE,
performances = TRUE, error.fun = NULL)
```

Arguments

random	if an integer value is specified, random parameter vectors are drawn from the parameter space.
nrepeat	specifies how often training shall be repeated.
repeat.aggregate	function for aggregating the repeated training results.
sampling	sampling scheme. If sampling = "cross", a cross-times cross validation is performed. If sampling = "boot", nboot training sets of size boot.size (part) are sampled (with replacement) from the supplied data. If sampling = "fix", a single split into training/validation set is used, the training set containing a fix part of the supplied data. Note that a separate validation set can be supplied via validation.x and validation.y. It is only used for sampling = "boot" and sampling = "fix"; in the latter case, fix is set to 1.
sampling.aggregate, sampling.dispersion	functions for aggregating the training results on the generated training samples (default: mean and standard deviation).
cross	number of partitions for cross-validation.
fix	part of the data used for training in fixed sampling.
nboot	number of bootstrap replications.
boot.size	size of the bootstrap samples.
best.model	if TRUE, the best model is trained and returned (the best parameter set is used for training on the complete training set).
performances	if TRUE, the performance results for all parameter combinations are returned.
error.fun	function returning the error measure to be minimized. It takes two arguments: a vector of true values and a vector of predicted values. If NULL, the misclassification error is used for categorical predictions and the mean squared error for numeric predictions.

Value

An object of class "tune.control" containing all the above parameters (either the defaults or the user specified values).

Author(s)

David Meyer

See Also

tune_wsvm, tune (in package **e1071**)

---

tune_wsvm *Parameter Tuning of Functions Using Grid Search*

Description

This generic function tunes hyperparameters of statistical methods using a grid search over supplied parameter ranges.

Usage

```
tune_wsvm(train.x, train.y = NULL, weight, use_zero_weight = FALSE,
     pre.check = TRUE, data = list(), validation.x = NULL,
     validation.y = NULL, validation.weight = NULL,
     weigthed.error = TRUE, ranges = NULL, predict.func = predict,
     tunecontrol = tune.control(), ...)
best.tune_wsvm(...)
```

Arguments

train.x	either a formula or a '*design* matrix' of predictors.
train.y	the response variable if train.x is a predictor matrix. Ignored if train.x is a formula.
weight	the weight of each subject. It should be in the same length of train.y.
use_zero_weight	if FALSE, any subjects in the training data and the validation data (if exist) with zero (or negative) weights will be removed.
pre.check	if TRUE, we prefit the model with partitioned training data using the first set of parameters in range. If fails (i.e., too many zero weight subjects in the partitioned training data), we re-partition the data and re-try the model for up to 10 times. This is useful when use_zero_weight=TRUE and there many zero weights subjects in the data.
data	data, if a formula interface is used. Ignored, if predictor matrix and response are supplied directly.
validation.x	an optional validation set. Depending on whether a formula interface is used or not, the response can be included in validation.x or separately specified using validation.y. Only used for bootstrap and fixed validation set (see tune.control)
validation.y	if no formula interface is used, the response of the (optional) validation set. Only used for bootstrap and fixed validation set (see tune.control)
validation.weight	the weight of each subject in the validation set. Will be set to 1, if the user does not provide.

weigthed.error	if TRUE, the preformance measure will be weighted.
ranges	a named list of parameter vectors spanning the sampling space. See wsvm. The vectors will usually be created by seq.
predict.func	optional predict function, if the standard predict behavior is inadequate.
tunecontrol	object of class "tune.control", as created by the function tune.control(). In additon, tune.control$error.fun should be a function that takes three arguments: (true y, predicted y, weight). If omitted, tune.control() gives the defaults.
...	Further parameters passed to the training functions.

Details

As performance measure, the classification error is used for classification, and the mean squared error for regression. It is possible to specify only one parameter combination (i.e., vectors of length 1) to obtain an error estimation of the specified type (bootstrap, cross-classification, etc.) on the given data set.

Cross-validation randomizes the data set before building the splits which—once created—remain constant during the training process. The splits can be recovered through the train.ind component of the returned object.

Value

For tune_wsvm, an object of class tune_wsvm, including the components:

best.parameters	a 1 x k data frame, k number of parameters.
best.performance	best achieved performance.
performances	if requested, a data frame of all parameter combinations along with the corresponding performance results.
train.ind	list of index vectors used for splits into training and validation sets.
best.model	if requested, the model trained on the complete training data using the best parameter combination.

best.tune_wsvm() returns the best model detected by tune_wsvm.

Author(s)

David Meyer
Modified by Tianchen Xu <tx2155@columbia.edu>

See Also

tune.control, plot.tune_wsvm

Examples

```
data(iris)

obj <- tune_wsvm(Species~., weight = c(rep(0.8, 50),rep(1,100)),
            data = iris, ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
            tunecontrol = tune.control(sampling = "fix"))


set.seed(11)
obj <- tune_wsvm(Species~., weight = c(rep(0, 52),rep(1,98)),
            data = iris, use_zero_weight = TRUE,
            ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
            tunecontrol = tune.control(sampling = "bootstrap"))



summary(obj)
plot(obj, transform.x = log2, transform.y = log2)
plot(obj, type = "perspective", theta = 120, phi = 45)

best.tune_wsvm(Species~.,weight = c(rep(0.08, 50),rep(1,100)),
            data = iris, ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
            tunecontrol = tune.control(sampling = "fix"))
```

---

wsvm *Subject Weighted Support Vector Machines*

Description

wsvm is used to train a subject weighted support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
wsvm(formula, weight, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
wsvm(x, y = NULL, weight, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 100, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Arguments

formula	a symbolic description of the model to be fit.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'wsvm' is called from.
x	a data matrix, a vector, or a sparse '*design* matrix' (object of class Matrix provided by the **Matrix** package, or of class matrix.csr provided by the **SparseM** package, or of class simple_triplet_matrix provided by the **slam** package). Or a kernel matrix of class kernelMatrix by the **kernlab** package.
y	a response vector with one label for each row/component of x. Can be either a factor (for classification tasks) or a numeric vector (for regression).
weight	the weight of each subject. It should be in the same length of y.
scale	A logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. By default, data are scaled internally (both x and y variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions. *If x is a design matrix which contains dummy variables, please make these variable NOT scaled.*
type	wsvm can be used as a classification machine, as a regression machine, or for novelty detection. Depending of whether y is a factor or not, the default setting for type is C-classification or eps-regression, respectively, but may be overwritten by setting an explicit value. Valid options are: • C-classification • nu-classification • one-classification (for novelty detection) • eps-regression • nu-regression
kernel	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. **linear:** $u'v$ **polynomial:** $(\gamma u'v + coef0)^{degree}$ **radial basis:** $e^(- \gamma \lvert u - v \rvert^2)$ **sigmoid:** $tanh(\gamma u'v + coef0)$ **precomputed:** x is a precomputed kernel matrix that contains NO missing values. scale will not work. Cannot use subset and na.action with this kernel.
degree	parameter needed for kernel of type polynomial (default: 3)
gamma	parameter needed for all kernels except linear (default: 1/(data dimension))
coef0	parameter needed for kernels of type polynomial and sigmoid (default: 0)
cost	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
nu	parameter needed for nu-classification, nu-regression, and one-classification

class.weights	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named. Specifying "inverse" will choose the weights *inversely* proportional to the class distribution.
cachesize	cache memory in MB (default 100)
tolerance	tolerance of termination criterion (default: 0.001)
epsilon	epsilon in the insensitive-loss function (default: 0.1)
shrinking	option whether to use the shrinking-heuristics (default: TRUE)
cross	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression. Note the result is not weighted. For weighted results, use tune_wsvm fucntion.
fitted	logical indicating whether the fitted values should be computed and included in the model or not (default: TRUE)
probability	logical indicating whether the model should allow for probability predictions.
...	additional parameters for the low level fitting function wsvm.default
subset	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
na.action	A function to specify the action to be taken if NAs are found. The default action is na.omit, which leads to rejection of cases with missing values on any required variable. An alternative is na.fail, which causes an error if NA cases are found. (NOTE: If given, this argument must be named.)

Details

The original libsvm does not support subject/instance weighted svm. From the 'LIBSVM Tools' https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#weights_for_data_instances, we are able to use a modified version of libsvm to support subject weights.

For multiclass-classification with k levels, k>2, libsvm uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme.

libsvm internally uses a sparse data representation, which is also high-level supported by the package **SparseM**.

If the predictor variables include factors, the formula interface must be used to get a correct model matrix or make x a design matrix.

When using the formula interface and na.action is na.omit, we delete any subjects with missing values on x, y (if exists) or weight in the training and predicting procedure (when fitted = TRUE). When using the x, y interface and na.action is na.omit, we delete any subjects with missing values on x, y (if exists) or weight in the training procedure, and retain the subjects with missing values only on weight in the predicting procedure (when fitted = TRUE).

plot.wsvm allows a simple graphical visualization of classification models.

The probability model for classification fits a logistic distribution using maximum likelihood to the decision values of all binary classifiers, and computes the a-posteriori class probabilities for the multi-class problem using quadratic optimization. The probabilistic regression model assumes

(zero-mean) laplace-distributed errors for the predictions, and estimates the scale parameter using maximum likelihood.

For linear kernel, the coefficients of the regression/decision hyperplane can be extracted using the `coef` method (see examples).

Value

An object of class `"wsvm"` containing the fitted model, including:

`SV`	The resulting support vectors (possibly scaled).
`index`	The index of the resulting support vectors in the data matrix. Note that this index refers to the preprocessed data (after the possible effect of `na.omit` and `subset`)
`coefs`	The corresponding coefficients times the training labels.
`rho`	The negative intercept.
`sigma`	In case of a probabilistic regression model, the scale parameter of the hypothesized (zero-mean) laplace distribution estimated by maximum likelihood.
`probA`, `probB`	numeric vectors of length k(k-1)/2, k number of classes, containing the parameters of the logistic distributions fitted to the decision values of the binary classifiers (1 / (1 + exp(a x + b))).

Note

Data are scaled internally, usually yielding better results.

Parameters of SVM-models usually *must* be tuned to yield sensible results!

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
Modified by Tianchen Xu <tx2155@columbia.edu>

References

- Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Ming-Wei Chang, Hsuan-Tien Lin, Ming-Hen Tsai, Chia-Hua Ho and Hsiang-Fu Yu
Weights for data instances
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#weights_for_data_instances
- Exact formulations of models, algorithms, etc. can be found in the document:
Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz
- More implementation details and speed benchmarks can be found on: Rong-En Fan and Pai-Hsune Chen and Chih-Jen Lin:
Working Set Selection Using the Second Order Information for Training SVM
https://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf

See Also

predict.wsvm, plot.wsvm, tune_wsvm, matrix.csr (in package **SparseM**)

Examples

```
## check what is loaded
dllpath <- getLoadedDLLs()
getDLLRegisteredRoutines(dllpath$WeightSVM[[2]])

## load dataset
data(iris)

## classification mode
# default with factor response:
model1 <- wsvm(Species ~ ., weight = rep(1,150), data = iris) # same weights
model2 <- wsvm(x = iris[,1:4], y = iris[,5],
              weight = c(rep(0.08, 50),rep(1,100))) # less weights to setosa
# alternatively the traditional interface:
x <- subset(iris, select = -Species)
y <- iris$Species
model3 <- wsvm(x, y, weight = rep(10,150)) # similar to model 1,
                                # but larger weights for all subjects

# These models provide error/warning info
try(wsvm(x, y)) # no weight
try(wsvm(x, y, weight = rep(10,100))) # wrong length
try(wsvm(x, y, weight = c(Inf, rep(1,149)))) # contains inf weight

print(model1)
summary(model1)

# test with train data
pred <- predict(model1, iris[,1:4])
# (same as:)
pred <- fitted(model1)

# Check accuracy:
table(pred, y) # model 1, equal weights

# compute decision values and probabilities:
pred <- predict(model1, x, decision.values = TRUE)
attr(pred, "decision.values")[1:4,]

# visualize (classes by color, SV by crosses):
plot(cmdscale(dist(iris[,-5])),
     col = as.integer(iris[,5]),
     pch = c("o","+")[1:150 %in% model1$index + 1]) # model 1
plot(cmdscale(dist(iris[,-5])),
     col = as.integer(iris[,5]),
     pch = c("o","+")[1:150 %in% model2$index + 1])
  # In model 2, less support vectors are based on setosa
```

```
## try regression mode on two dimensions
# create data
x <- seq(0.1, 5, by = 0.05)
y <- log(x) + rnorm(x, sd = 0.2)

# estimate model and predict input values
model1 <- wsvm(x, y, weight = rep(1,99))
model2 <- wsvm(x, y, weight = seq(99,1,length.out = 99)) # decreasing weights
# library(kernlab)
# model3 <- wsvm(kernlab::kernelMatrix(kernlab::rbfdot(sigma = 1), x), y,
#      weight = rep(1,99), kernel = 'precomputed') # try user defined kernel

# visualize
plot(x, y)
lines(x, log(x), col = 2)
points(x, fitted(model1), col = 4)
points(x, fitted(model2), col = 3) # better fit for the first few points
# points(x, fitted(model3), col = 5) # similar to model 1 with user defined kernel

## density-estimation
# create 2-dim. normal with rho=0:
X <- data.frame(a = rnorm(1000), b = rnorm(1000))
attach(X)

# formula interface:
model <- wsvm(~ a + b, gamma = 0.1, weight = c(seq(5000,1,length.out = 500),1:500))

# test:
newdata <- data.frame(a = c(0, 4), b = c(0, 4))

# visualize:
plot(X, col = 1:1000 %in% model$index + 1, xlim = c(-5,5), ylim=c(-5,5))
points(newdata, pch = "+", col = 2, cex = 5)

## class weights:
i2 <- iris
levels(i2$Species)[3] <- "versicolor"
summary(i2$Species)
wts <- 100 / table(i2$Species)
wts
m <- wsvm(Species ~ ., data = i2, class.weights = wts, weight=rep(1,150))

## extract coefficients for linear kernel

# a. regression
x <- 1:100
y <- x + rnorm(100)
m <- wsvm(y ~ x, scale = FALSE, kernel = "linear", weight = rep(1,100))
coef(m)
plot(y ~ x)
abline(m, col = "red")
```

```
# b. classification
# transform iris data to binary problem, and scale data
setosa <- as.factor(iris$Species == "setosa")
iris2 = scale(iris[,-5])

# fit binary C-classification model
model1 <- wsvm(setosa ~ Petal.Width + Petal.Length,
          data = iris2, kernel = "linear", weight = rep(1,150))
model2 <- wsvm(setosa ~ Petal.Width + Petal.Length,
               data = iris2, kernel = "linear",
               weight = c(rep(0.08, 50),rep(1,100))) # less weights to setosa

# plot data and separating hyperplane
plot(Petal.Length ~ Petal.Width, data = iris2, col = setosa)
(cf <- coef(model1))
abline(-cf[1]/cf[3], -cf[2]/cf[3], col = "red")
(cf2 <- coef(model2))
abline(-cf2[1]/cf2[3], -cf2[2]/cf2[3], col = "red", lty = 2)

# plot margin and mark support vectors
abline(-(cf[1] + 1)/cf[3], -cf[2]/cf[3], col = "blue")
abline(-(cf[1] - 1)/cf[3], -cf[2]/cf[3], col = "blue")
points(model1$SV, pch = 5, cex = 2)
abline(-(cf2[1] + 1)/cf2[3], -cf2[2]/cf2[3], col = "blue", lty = 2)
abline(-(cf2[1] - 1)/cf2[3], -cf2[2]/cf2[3], col = "blue", lty = 2)
points(model2$SV, pch = 6, cex = 2)
```

Index