

# svm() internals

Some technical notes about the svm() in package e1071

by David Meyer
FH Technikum Wien, Austria
David.Meyer@R-Project.org

October 13, 2020

This document explains how to use the parameters in an object returned by svm() for own prediction functions.

## 1 Binary Classifier

For class prediction in the binary case, the class of a new data vector \( n \) is usually given by the sign of

\[
\sum_i a_i y_i K(x_i, n) + \rho \tag{1}
\]

where \( x_i \) is the \( i \)-th support vector, \( y_i \) the corresponding label, \( a_i \) the corresponding coefficient, and \( K \) is the kernel (for example the linear one, i.e. \( K(u, v) = u^T v \)).

Now, the libsvm library interfaced by the svm() function actually returns \( a_i y_i \) as \( i \)-th coefficient and the negative \( \rho \), so in fact uses the formula:

\[
\sum_i \text{coef}_i K(x_i, n) - \rho
\]

where the training examples (=training data) are labeled \{1,-1\} (!). A simplified R function for prediction with linear kernel would be:

```r
svmpred <- function (m, newdata, K=crossprod) {
  ## this guy does the computation:
  pred.one <- function (x)
    sign(sum(sapply(1:m$tot.nSV, function (j)
                        K(m$SV[j,], x) * m$coefs[j]
                    ) - m$rho

  ## this is just for convenience:
  if (is.vector(newdata))
    newdata <- t(as.matrix(x))
  sapply (1:nrow(newdata),
    function (i) pred.one(newdata[i,]))
}
```

1

where `pred.one()` does the actual prediction for one new data vector, the remainder is just a convenience for prediction of multiple new examples. It is easy to extend this to other kernels, just replace `K()` with the appropriate function (see the help page for the formulas used) and supply the additional constants.

As we will see in the next section, the multi-class prediction is more complicated, because the coefficients of the diverse binary SVMs are stored in a compressed format.

## 2 Multiclass-classifier

To handle $k$ classes, $k > 2$, `svm()` trains all binary subclassifiers (one-against-one-method) and then uses a voting mechanism to determine the actual class. Now, this means $k(k-1)/2$ classifiers, hence in principle $k(k-1)/2$ sets of SVs, coefficients and rhos. These are stored in a compressed format:

1. Only one SV is stored in case it were used by several classifiers. The `model$SV-matrix` is ordered by classes, and you find the starting indices by using `nSV` (number of SVs):

   ```
   start <- c(1, cumsum(model$nSV))
   start <- start[-length(start)]
   ```

   `sum(nSV)` equals the total number of (distinct) SVs.

2. The coefficients of the SVs are stored in the `model$coefs-matrix`, grouped by classes. Because the separating hyperplanes found by the SVM algorithm has SVs on both sides, you will have two sets of coefficients per binary classifier, and e.g., for 3 classes, you could build a **block-matrix** like this for the classifiers $(i, j)$, where $i, j = $class numbers:

   ```
   i \ j | 0 | 1 | 2
   0 | X | set (0, 1) | set (0, 2)
   1 | set (1, 0) | X | set (1, 2)
   2 | set (2, 0) | set (2, 1) | X
   ```

   where `set(i, j)` are the coefficients for the classifier $(i, j)$, lying on the side of class $j$. Because there are no entries for $(i, i)$, we can save the diagonal and shift up the lower triangular matrix to get

   ```
   i \ j | 0 | 1 | 2
   0 | set (1,0) | set (0,1) | set (0,2)
   1 | set (2,0) | set (2,1) | set (1,2)
   ```

   Each set $(., j)$ has length `nSV[j]`, so of course, there will be some filling 0s in some sets.

   `model$coefs` is the **transposed** of such a matrix, therefore for a data set with, say, 6 classes, you get 6-1=5 columns.

   The coefficients of $(i, j)$ start at `model$coefs[start[i],j]` and those of $(j, i)$ at `model$coefs[start[j],i-1].`

3. The $k(k-1)/2$ rhos are just linearly stored in the vector `model$rho`.

---

2
The following code shows how to use this for prediction:

```r
## Linear Kernel function
K \left< function(i,j) \right> \text{crossprod}(i,j)

predsvm \left< function(object, newdata) \right>\
{\}
## compute start-index
start \left< c(1, \text{cumsum}(object$nSV)+1) \right>
start \left< start[-\text{length}(start)] \right>

## compute kernel values
kernel \left< \text{apply} (i:\text{object$tot.nSV}, \right.
\left. function (x) K(object$SV[x,], newdata)) \right>

## compute raw prediction for classifier (i,j)
predone \left< function (i,j) \right>\
{\}
## ranges for class i and j:
ri \left< start[i] : (start[i] + object$nSV[i] - 1) \right>
rj \left< start[j] : (start[j] + object$nSV[j] - 1) \right>

## coefs for (i,j):
coef1 \left< object$coefs[ri, j-1] \right>
coef2 \left< object$coefs[rj, i] \right>

## return raw values:
crossprod(coef1, kernel[ri]) + crossprod(coef2, kernel[rj])
}

## compute votes for all classifiers
evotes \left< \text{rep}(0, object$nclasses) \right>
c <- 0 # rho counter
for (i in 1 : (object$nclasses - 1))
for (j in (i + 1) : object$nclasses)
if (predone(i,j) > object$rho[c <- c + 1])
votes[i] <- votes[i] + 1
else
vvotes[j] <- votes[j] + 1

## return winner (index with max. votes)
object$levels[\text{which}(votes \%in\% \text{max}(votes))[1]]
}

In case data were scaled prior fitting the model (note that this is the default for \text{svm}()), the
new data needs to be scaled as well before applying the predicion functions, for example using the
following code snipped (object is an object returned by \text{svm}(), newdata a data frame):

```r
if (any(object$scaled))
newdata[,object$scaled] <-
scale(newdata[,object$scaled, drop = FALSE],
      center = object$x.scale$scaled:center",
      scale = object$x.scale$scaled:scale"
"
```

For regression, the response needs to be scaled as well before training, and the predictions need
to be scaled back accordingly.