svm() internals
Some technical notes about the svm() in package e1071

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

December 5, 2023

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

\[
\sum_i a_i y_i K(x_i, n) + \rho
\]

(1)

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

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
svm.pred <- 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$coeffs[j]
                ) - m$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,]))
}
```

This function can be used to make predictions for new data based on the support vectors and coefficients returned by the svm() function.
where \texttt{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 \texttt{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, \( k > 2 \), \texttt{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 \texttt{model$SV\_matrix} is ordered by classes, and you find the starting indices by using \texttt{nSV} (number of SVs):

   \[
   \begin{align*}
   \text{start} &\leftarrow c(1, \text{cumsum(model$nSV))}
   
   \text{start} &\leftarrow \text{start[length(start)]}
   
   \end{align*}
   \]

   \texttt{sum(nSV)} equals the total number of (distinct) SVs.

2. The coefficients of the SVs are stored in the \texttt{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 \((i, j)\) \((i, j=\text{class numbers})\):

   \[
   \begin{array}{ccc}
   1 \backslash j & 0 & 1 & 2 \\
   0 & X & \text{set (0, 1)} & \text{set (0, 2)} \\
   1 & \text{set (1, 0)} & X & \text{set (1, 2)} \\
   2 & \text{set (2, 0)} & \text{set (2, 1)} & X \\
   \end{array}
   \]

   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

   \[
   \begin{array}{ccc}
   i \backslash j & 0 & 1 & 2 \\
   0 & \text{set (1, 0)} & \text{set (0, 1)} & \text{set (0, 2)} \\
   1 & \text{set (2, 0)} & \text{set (2, 1)} & \text{set (1, 2)} \\
   \end{array}
   \]

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

   \texttt{model$coefs} is the \textit{transposed} of such a matrix, therefore for a data set with, say, \texttt{6} classes, you get 6-1=5 columns.

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

3. The \( k(k-1)/2 \) rhos are just linearly stored in the vector \texttt{model$rho}.
The following code shows how to use this for prediction:

```r
### Linear Kernel function
K <- function(i,j) crossprod(i,j)

predsvm <- function(object, newdata)
{
  ## compute start-index
  start <- c(1, cumsum(object$nsV)+1)
  start <- start[-length(start)]

  ## compute kernel values
  kernel <- sapply (1:object$tot.nSV,
                   function (x) K(object$SV[x,], newdata))

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

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

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

  ## compute votes for all classifiers
  votes <- rep(0,object$nclasses)
  c <- 0 # rho counter
  for (i in 1 : (object$nclasses - 1))
    for (j in (i + 1) : object$nclasses)
      if (predone(i,j) > object$rho[c < c + 1])
        votes[i] <- votes[i] + 1
      else
        votes[j] <- votes[j] + 1

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

In case data were scaled prior fitting the model (note that this is the default for `svm()`), the new data needs to be scaled as well before applying the prediction functions, for example using the following code snippet (object is an object returned by `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.