Summary on Support Vector Machines with Applications

Liu Huihang

2020.1.6

Support Vector Classifier

Assume our training data consists of N pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. Define a hyperplane by

$$\{x : f(x) = x^T \beta + \beta_0 = 0\} \tag{1}$$

where β is a unit vector: $\|\beta\| = 1$. A classification rule induced by $f(x)$ is

$$G(x) = \text{sign}\left[x^T \beta + \beta_0\right] \tag{2}$$

We know that $f(x)$ in (1) gives the signed distance from a point x to the hyperplane $f(x) = x^T \beta + \beta_0 = 0$. Since the classes are separable, we can find a function $f(x) = x^T \beta + \beta_0 = 0$ with $y_i f(x_i) > 0$, $\forall i$. Hence we are able to find the hyperplane that creates the biggest margin between the training points for class 1 and -1. The optimization problem

$$\begin{gathered} \max_{\beta, \beta_0, \|\beta\|=1} M \\ \text{subject to } y_i\left(x_i^T \beta + \beta_0\right) \geq M, i = 1, \ldots, N \end{gathered} \tag{3}$$

captures this concept. The band in the figure is M units away from the hyperplane on either side, and hence $2M$ units wide. It is called the margin.

Computations can be done by solving a quadratic programming problem with Lagrange multipliers method. please see *The Element of Statistical Learning* for detials.

The support vector classifier finds linear boundaries in the input feature space. As with other linear methods, we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Generally linear boundaries in the enlarged space achieve better training-class separation, and translate to nonlinear boundaries in the original space. Once the basis functions $h_m(x)$, $m = 1, \ldots, M$ are selected, the procedure is the same as before. We fit the SV classifier using input features $h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_M(x_i)), i = 1, \ldots, N$, and produce the (nonlinear) function $\hat{f}(x) = h(x)^T \hat{\beta} + \hat{\beta}_0$. The classifier is $\hat{G}(x) = sign(\hat{f}(x))$ as before.

The support vector machine classifier is an extension of this idea, where the dimension of the enlarged space is allowed to get very large, infinite in some cases. It might seem that the computations would become prohibitive. It would also seem that with sufficient basis functions, the data would be separable, and overfitting would occur. We first explore how the SVM technology deals with these issues. We then see that in fact the SVM classifier is solving a function-fitting problem using a particular criterion and form of regularization, and is part of a much bigger class of problems that includes the smoothing splines.

Real data: cats

The `cats` data set in the `MASS` package contains three variables, where the variable *Sex* indicates the gender of the cat ('F'or'M'), *Bwt* indicates the cat's weight (unit: kg), and *Hwt* indicates the cat's heart weight (unit: g). Our goal is to use the SVM algorithm to classify cats through two variables, *Bwt* and *Hwt*.

```r
# load package and data
library("MASS")
head(cats)
```

```
##   Sex Bwt Hwt
## 1   F 2.0 7.0
## 2   F 2.0 7.4
## 3   F 2.0 9.5
## 4   F 2.1 7.2
## 5   F 2.1 7.3
## 6   F 2.1 7.6
```

```r
# Select training samples (70%) and test samples (30%)
index <- sample(2,nrow(cats),replace = TRUE,prob=c(0.7,0.3))
traindata <- cats[index==1,]
testdata <- cats[index==2,]

# model
cats_svm_model <- svm(Sex~.,data=traindata)
print(cats_svm_model)
```

```
##
## Call:
## svm(formula = Sex ~ ., data = traindata)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  61
```

```r
# check result on training data
cats_svm_model_pred_1 <- predict(cats_svm_model,traindata[,-1])
cats_table_1 <- table(pred=cats_svm_model_pred_1,true=traindata[,1])
print(cats_table_1)
```

```
##     true
## pred  F  M
##    F 23 11
##    M  9 54
```

```r
accuracy_1 <- sum(diag(cats_table_1))/sum(cats_table_1)
print(accuracy_1)
```

```
## [1] 0.7938144
```

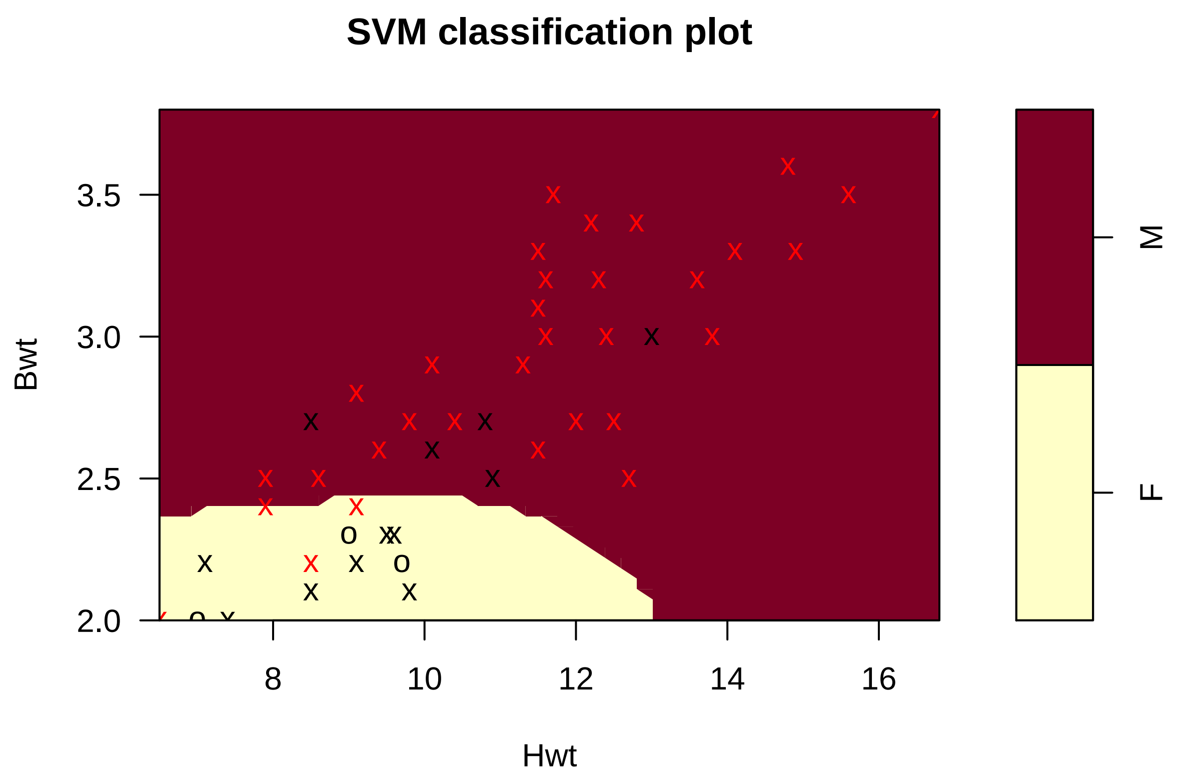
```r
# check on the test data
cats_svm_model_pred_2 <- predict(cats_svm_model,testdata[,-1])
cats_table_2 <- table(pred=cats_svm_model_pred_2,true=testdata[,1])
cats_table_2
```

```
##     true
## pred  F  M
##    F 10  3
##    M  5 29
```

```
accuracy_2 <- sum(diag(cats_table_2))/sum(cats_table_2)
accuracy_2
```

```
## [1] 0.8297872
```

```
# plot the result
plot(cats_svm_model,testdata)
```



Real data: iris

First of all, for the classification problem, the 'type' parameter in the `svm()` han function has three options: *C-classification*, *nu-classification*, and *one-classification*. And the 'kernel' parameter has four options: *linear*, *polynomial*, *radial* and *sigmoid*.

In order to comprehensively compare the model effects of these *12* combinations, build a self-written function

```
svm_test <- function(x,y){
  type <- c('C-classification','nu-classification','one-classification')
  kernel <- c('linear','polynomial','radial','sigmoid')
  pred <- array(0, dim=c(nrow(x),3,4))
  errors <- matrix(0,3,4)
  dimnames(errors) <- list(type, kernel)
  for(i in 1:3){
    for(j in 1:4){
      pred[,i,j] <- predict(object = svm(x, y, type = type[i], kernel = kernel[j]), newdata = x)
      if(i > 2) errors[i,j] <- sum(pred[,i,j] != 1)
      else errors[i,j] <- sum(pred[,i,j] != as.integer(y))
      }
    }
  return(errors)
}

iris_x <- iris[,1:4]
iris_y <- iris[,5]
svm_test(x=iris_x,y=iris_y)
```

```
##                     linear polynomial radial sigmoid
## C-classification         5          7      4      17
## nu-classification        5         14      5      12
## one-classification     102         75     76      75
```

According to the above results, type = 'C-classification' and kernel = 'radial' return the smallest amount of false positives, so this combination can be considered to classify IRIS.

```
iris_model <- svm(x=iris[,1:4],y=iris[,5],type = 'C-classification', kernel = 'radial')
iris_pred <- predict(object = iris_model, newdata = iris[,1:4])
iris_Freq <- table(iris[,5], iris_pred)
print(iris_Freq)
```

```
##               iris_pred
##                setosa versicolor virginica
##   setosa           50          0         0
##   versicolor        0         48         2
##   virginica         0          2        48
```

In this way, we saw that the correct classification rate reached 97.3%, and 4 samples were misjudged. In order to further improve the classification effect, we can use the `tune.svm()` function to find the optimal parameters of the `svm()` function.