PHS 597 – Homework 4 – Support Vector Machine Implementation – Fall 2021

Havell Markus, hmm5304@psu.edu

November 28, 2021

Assginment description

- The goal of this assignment is to implement support vector machine (SVM) algorithm where the classes overlap in feature space.
- This is done by allowing some points to be on the wrong side of the margin.
- We define slack variables $\xi = (\xi_1, \xi_2, ..., \xi_N)$ and modify the constraint function of the linearly seperable SVM case as shown in Equation 1 with the constraints $\forall i, \xi_i \geq 0$, and $\sum_{i=1}^{N} \xi_i \leq C$ for some constant C.

$$y_i(x_i^T + \beta + \beta_0) \geq M(1 - \xi_i) \tag{1}$$

- The Lagrange (dual) objective function is shown in Equation 2. We maximize Equation 2 subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$.

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2}$$

- After solving for the Lagrange multipliers α_i using quadratic programming, the opimal hyperplane can be defined as Equation 3 and 4 for $\alpha_i > 0$:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{3}$$

$$\beta_0 = average_{i \in N}(y_i - w^T x_i) \tag{4}$$

Implementation

```
################################################################################
#### Load packages
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("data.table")
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

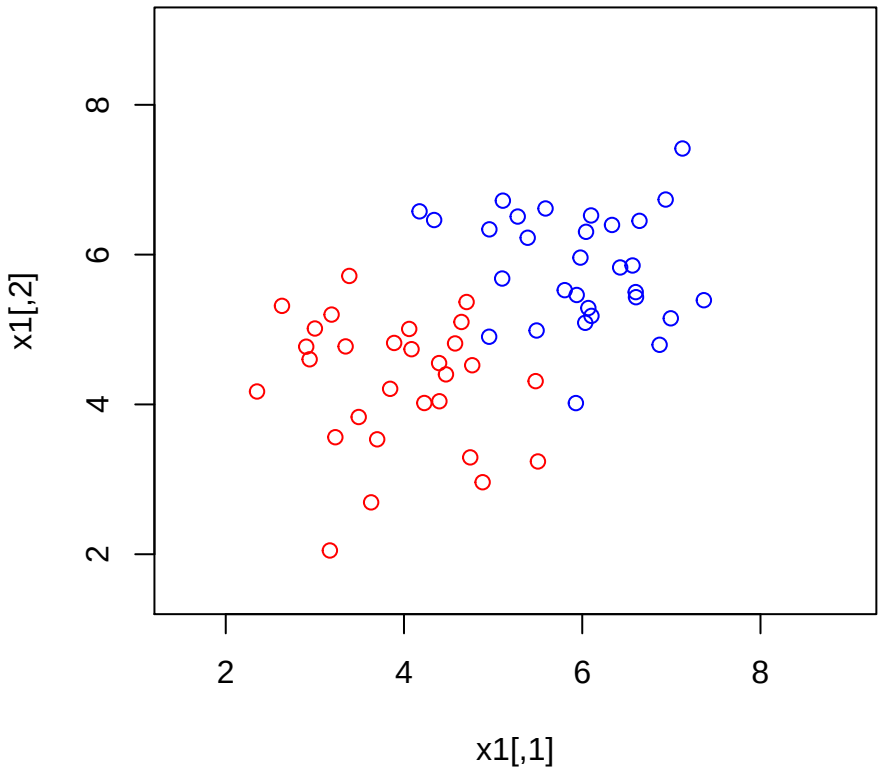
```
library("foreach")
library("ggplot2")
library("quadprog")
library("MASS")
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library("mvtnorm")
library("Matrix")
```

- data generation

```
#################################################################################
## drawing the scenario of a classification problem where linear boundaries can be
## expanded by adding quadratic terms of x;
set.seed(121)
x1=rmvnorm(30,mean=c(1.1,1.1),sigma=1*diag(2))
x2=rmvnorm(30,mean=c(3,3),sigma=0.8*diag(2))
x1=matrix(x1,ncol=2);
x2=matrix(x2,ncol=2);
y1=rep(1,nrow(x1));
y2=rep(-1,nrow(x2));
x1 <- x1+3
x2 <- x2+3
x=rbind(x1,x2);
y=c(y1,y2);
plot(x1,xlim=c(1.5,9),ylim=c(1.5,9),col='red');
points(x2,col='blue');
```



- Defining constraints for quadratic programming, in the form $min\frac{1}{2}x^TDx - d^Tx$ with the constraints $A^Tx \geq b$
  - $D = (xx^T) \times (yy^T)$
  - d is a vector of 1 with length of number of parameter
- The implementation allows to also solve for linearly seperable cases, by not defining c.

```
# Matrix appearing in the quadratic function
Dmat <- (x %*% t(x)) * (y %*% t(y))
pd_Dmat <- nearPD(Dmat)
pd_Dmat <- as.matrix(pd_Dmat$mat)

# Vector appearing in the quadratic function
dvec <- rep(1, length(y))

# Matrix defining the constraints

Amat <- t(rbind(y, diag(1, nrow = length(y)),
                diag(-1, nrow = length(y))))

# Vector holding the value of b_0
```

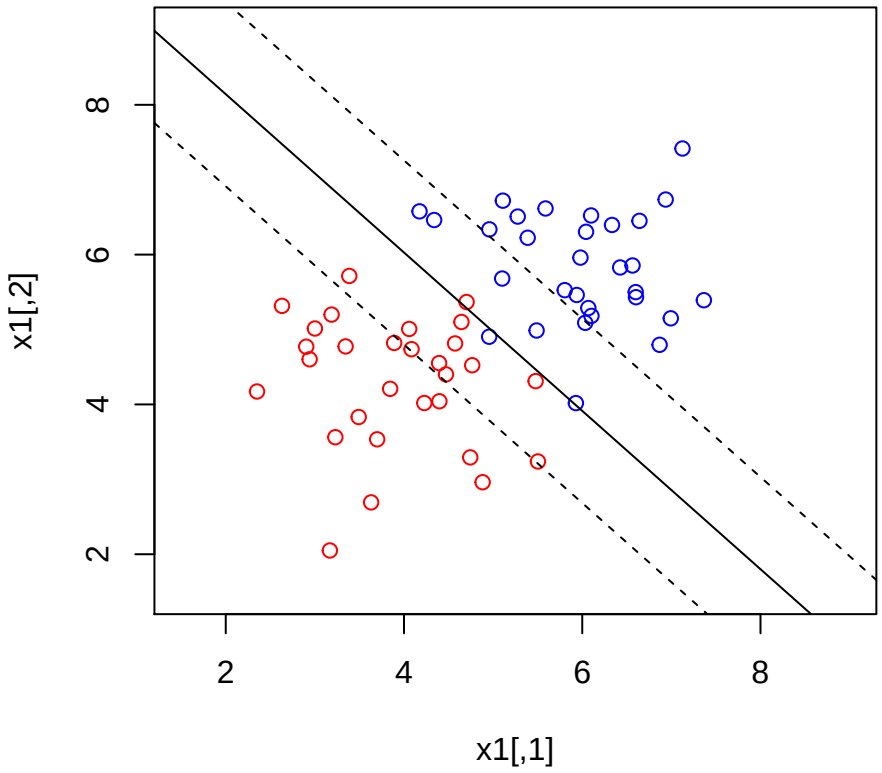
```
c <- 0.1
if (!is.null(C)) { # soft-margin
  Amat <- t(rbind(y, diag(1, nrow = length(y)),
                  diag(-1, nrow = length(y))))

  bvec <- c(0, rep(0, length(y)), rep(c*-1, length(y)))
} else {           # hard-margin
  Amat <- t(rbind(y, diag(1, nrow = length(y))))
  bvec <- c(0, rep(0, length(y)))
}

# meq indicates how many constraints are equality
# Only the first constraint is equality so meq = 1
qp <- solve.QP(pd_Dmat, dvec, Amat, bvec, meq = 1)
alphas <- qp$solution
```

- Fining the support vectors, solving for the w and β_0, and plotting the results.

```
## deifning support vectors
ind <- which(alphas > 4e-11)
ind
```

```
##  [1]  3  6  8 11 12 16 19 20 23 24 25 30 31 32 35 37 46 48 49 50 51 53 60
```

```
# ind <- c(3,32)

sv <- x[ind,]
sv_y <- y[ind]
sv_alpha <- alphas[ind]

w <- foreach(i = 1:length(ind), .combine = "rbind") %do% {
  sv_alpha[i] * sv_y[i] * sv[i,]
} %>% colSums()

b <- foreach(i = 1:length(ind), .combine = "c") %do% {
  as.numeric(sv_y[i] - (w %*% sv[i,]))
} %>% mean()

plot(x1,xlim=c(1.5,9),ylim=c(1.5,9),col='red');
points(x2,col='blue');

abline(a=(b/(-1*w[2])), b=w[1]/(-1*w[2]))
abline(a=(b/(-1*w[2]))+1.23, b=w[1]/(-1*w[2]), lty = 2)
abline(a=(b/(-1*w[2]))-1.23, b=w[1]/(-1*w[2]), lty = 2)
```




```
## beta_0
(b/(-1*w[2]))
```

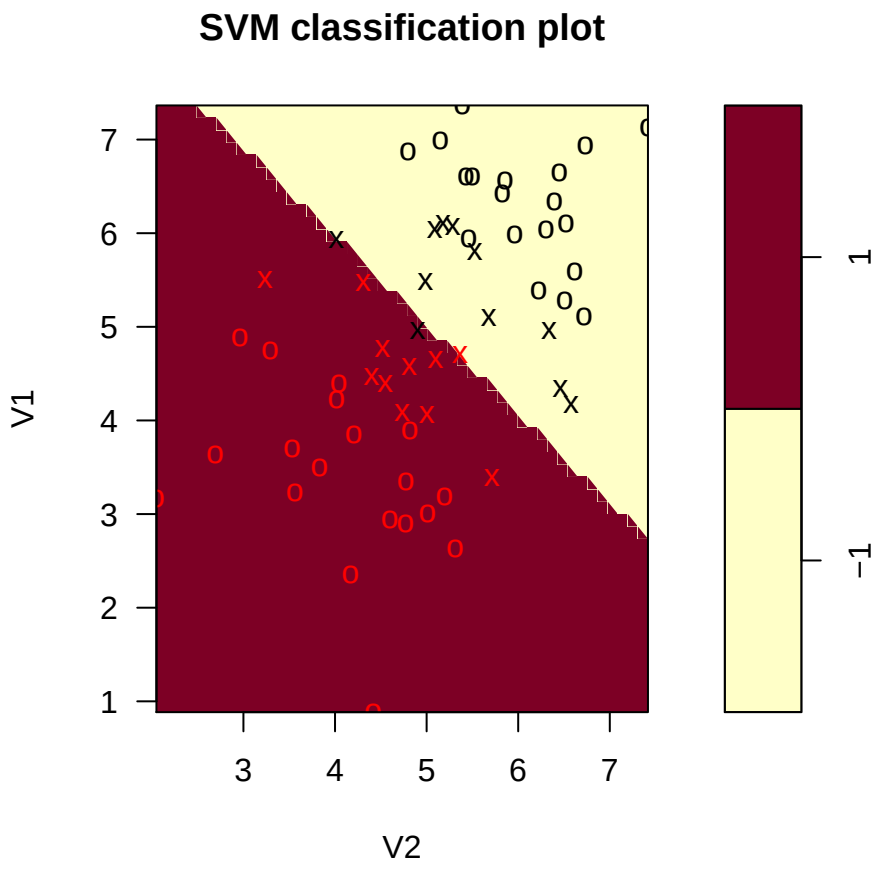
```
## [1] 10.25585
```

```
## weights
w[1]/(-1*w[2])
```

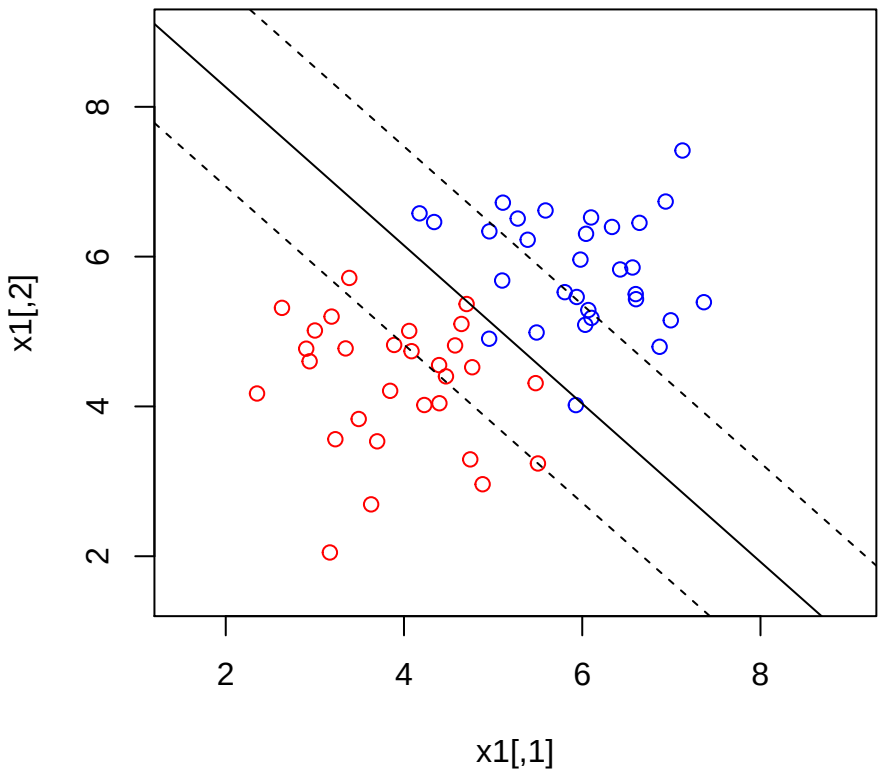
```
## [1] -1.056946
```

Results comparison

```
library("e1071")

data.train <- as.data.frame(x)
data.train$y <- y
data.train$y <- as.factor(data.train$y)
svm.model <- svm(y ~ ., data = data.train, cost = 0.1, gamma = 1, kernel = "linear", scale = FALSE)
plot(svm.model, data.train)
```




```
beta = drop(t(svm.model$coefs)%*%x[svm.model$index,])
beta0 = svm.model$rho

plot(x1,xlim=c(1.5,9),ylim=c(1.5,9),col='red');
points(x2,col='blue');

abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```




```
## beta_0
beta0 / beta[2]
```

```
## [1] 10.37384
```

```
## weights
-beta[1] / beta[2]
```

```
## [1] -1.056237
```

- The final results are very close, the implementation was successful
- Lastly, the c parameter can be used as a tuning parameter, and can be mined using cross validation