STAT 542: Homework 8

Sharvi Tomar (stomar2)

Table of Contents

About HW8

In this HW, we will code both the primal and dual form of SVM and utilize a general quadratic programming (quadprog package) solve to help us obtain the solution.

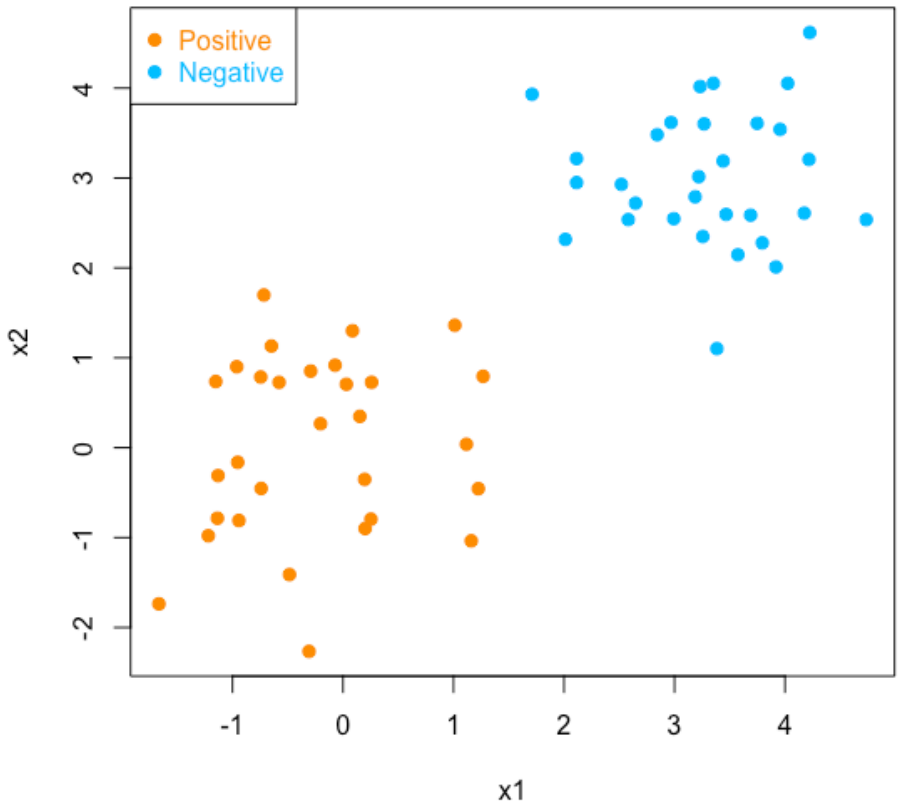
Question 1 [50 Points] Sovling SVM using Quadratic Programming

Install the quadprog package. The same package is also available in Python. However, make sure to read their documentations carefully. We will utilize the function solve.QP to solve SVM. This function is trying to perform the minimization problem:

where is the unknown parameter. For more details, read the documentation of the package on CRAN. Use our the provided training data. This is a linearly separable problem.

```
train = read.csv("SVM-Q1.csv")
x = as.matrix(train[, 1:2])
y = train[, 3]
```

```
plot(x, col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", ylab = "x2")
legend("topleft", c("Positive","Negative"), col=c("darkorange", "deepskyblue"),
       pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```



a) [25 points] The Primal Form

Use the formulation defined on page 13 of the SVM lecture note. The primal problem is

Perform the following:

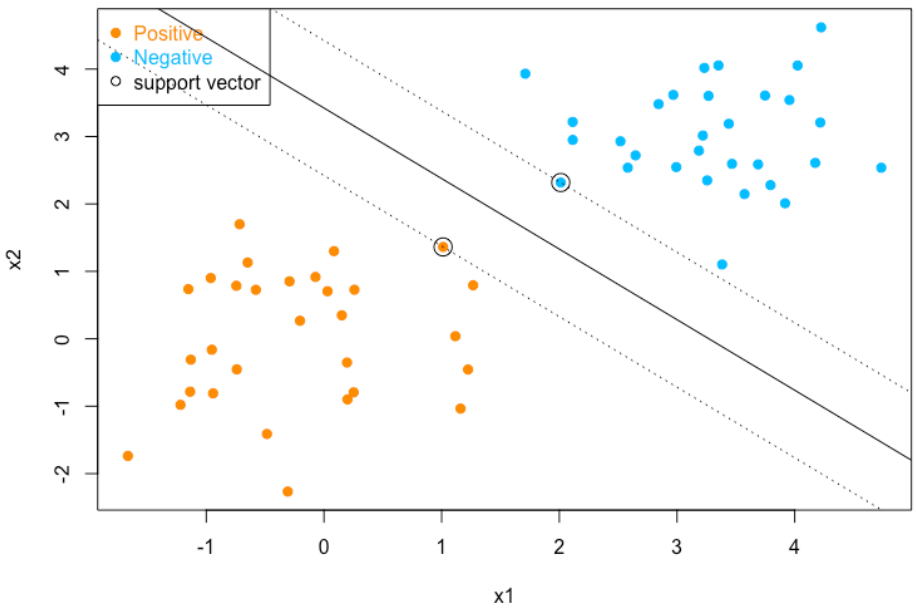
Let in the solve.QP() function. Properly define , , and corresponding to this for the linearly separable SVM primal problem.
Calculate the decision function by solving this optimization problem with the solve.QP() function.
Report our and
Plot the decision line on top the previous training data scatter plot. Include the two margin lines. Clearly mark the support vectors.

Note: The package requires to be positive definite, while it is not true in our case. To address this problem, **add to the top-left element** of your matrix, which is the one corresponding to . This will make invertible. This may affect your results slightly. So be careful when plotting your support vectors.

```
library(quadprog)
n = length(y)
# calculate D, d, A, b0
D = diag(3)
D[1,1] = 1e-10
d = rep(0, 3)
A = t(sweep(cbind(1, x), 1, y, FUN = "*"))
b0 = rep(1, n)

# solve the quadratic program problem
result = solve.QP(Dmat = D, dvec = d, Amat = A, bvec = b0)$solution
beta = result[2:3]
beta0 = result[1]

# plot the decision line and the support vectors
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", ylab = "x2")
points(x[sweep(x %*% beta + beta0, 1, y, FUN = "*") <= 1+1e-5, ], pch = 1, cex = 2)
legend("topleft", c("Positive","Negative", "support vector"),
    col=c("darkorange", "deepskyblue", "black"), pch=c(19, 19, 1),
    text.col=c("darkorange", "deepskyblue", "black"))
abline(-beta0/beta[2], -beta[1]/beta[2])
abline(-(beta0 - 1)/beta[2], -beta[1]/beta[2], lty=3)
abline(-(beta0 + 1)/beta[2], -beta[1]/beta[2], lty=3)
```



b) [25 points] The Dual Form

Formulate the SVM **dual** problem on page 21 the lecture note. The dual problem is

Perform the following:

Let . Then properly define , , and corresponding to this for our SVM problem.
Note: Equality constrains can be addressed using the meq argument.
Obtain the solution using the solve.QP() function, and convert the solution into and .

You need to report * A table including of both Q1a and Q1b. Only keep first three digits after the decimal point. * Plot the decision line on top of our scatter plot. Include the two margin lines. Clearly mark the support vectors. * Report the norm of , where and are the 3-dimensional solution obtained in Q1a and Q1b, respectively.

dimensional solution obtained in Q1a and Q1b, respectively.

Note: Again, may not be positive definite. This time, add to all diagonal elements to . This may affect your results slightly. So be careful when plotting your support vectors.

```
# calculate D,  d, A, b0
D = x %*% t(x) * y %*% t(y)
D = D + diag(1e-10, n) # add ridge
d = rep(1, n)
A = t(rbind(t(y), diag(1, n)))
b0 = rep(0, n+1)

# solve the quadratic program
alpha = solve.QP(D, d, A, b0, meq = 1)$solution

# beta in Q1a
beta_1a = c(beta0, beta)

# get beta for 1b
beta = t(x) %*% (alpha * y)
beta0 = -(max((x %*% beta)[y == -1])+ min((x %*% beta)[y == 1]))/2
beta_1b = c(beta0, beta)

beta_tab = rbind(beta_1a, beta_1b)
rownames(beta_tab) = c("Q1a", "Q1b")
colnames(beta_tab) = paste0("beta", 0:2)
knitr::kable(beta_tab, digits = 3,
          caption = "beta obtained from Q1a and Q1b")
```

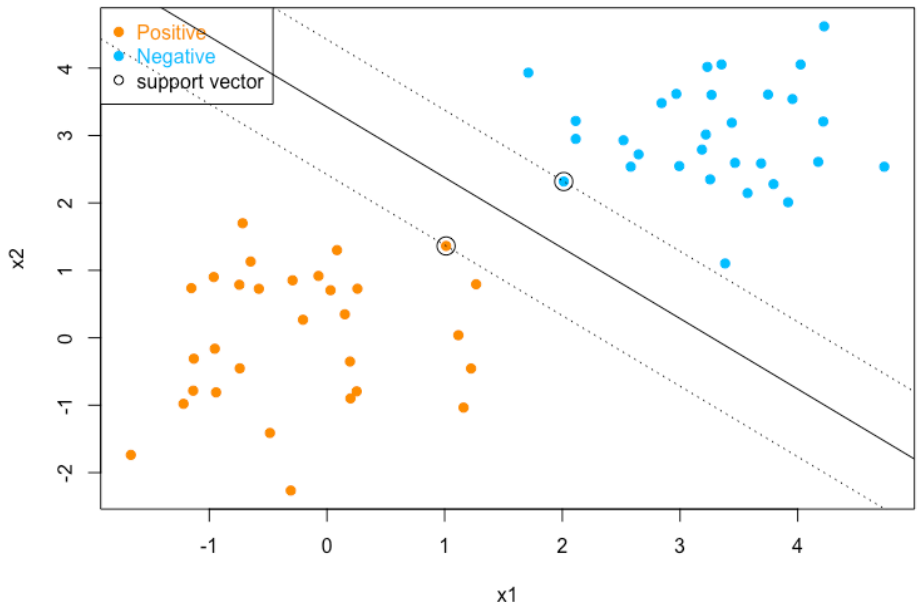
beta obtained from Q1a and Q1b

	beta0	beta1	beta2
Q1a	3.419	-1.046	-0.999
Q1b	3.419	-1.046	-0.999

```
# plot the decision line and the support vectors
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19,
   xlab = "x1", ylab = "x2")

# find the points with large enough (significantly non-zero) alpha
points(x[alpha > 0.1,], pch = 1, cex = 2)
legend("topleft", c("Positive","Negative", "support vector"),
     col=c("darkorange", "deepskyblue", "black"), pch=c(19, 19, 1),
     text.col=c("darkorange", "deepskyblue", "black"))

abline(-beta0/beta[2], -beta[1]/beta[2])
abline(-(beta0 - 1)/beta[2], -beta[1]/beta[2], lty=3)
abline(-(beta0 + 1)/beta[2], -beta[1]/beta[2], lty=3)
```




```
# L1 norm of beta in a and b
sum(abs(beta_tab[1, ] - beta_tab[2, ]))
## [1] 0.0001774678
```

Question 2 [20 Points] Linearly nonseparable SVM

In this question, we will follow the formulation in Page 30 to solve a linearly nonseparable SVM. The dual problem is given by

Perform the following:

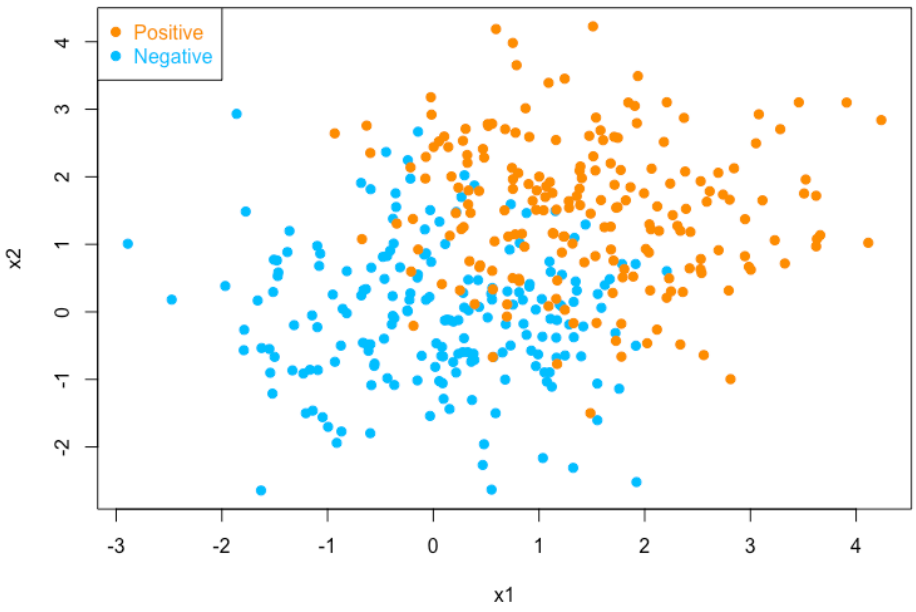
Let . Then properly define , , and corresponding to this for this problem. Use as the penalty team.
Note: Equality constrains can be addressed using the meq argument.
Obtain the solution using the solve.QP() function, and convert the solution into and . Note: use the information provided on page 32 to obtain the support vectors and .
Your solution may encounter numerical errors, e.g., very small negative values, or values very close to . You could consider thresholding them to exactly 0 or
Your may not be definite positive, so consider adding to its diagonal elements.

```
train = read.csv("SVM-Q2.csv")
x = as.matrix(train[, 1:2])
y = train[, 3]

set.seed(20)
n = 200 # number of data points for each class
p = 2 # dimension

# Generate the positive and negative examples
xpos <- matrix(rnorm(n*p,mean=0,sd=1),n,p)
xneg <- matrix(rnorm(n*p,mean=1.5,sd=1),n,p)
x <- rbind(xpos,xneg)
y <- c(rep(-1, n), rep(1, n))


plot(x, col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", ylab = "x2")
legend("topleft", c("Positive","Negative"), col=c("darkorange", "deepskyblue"),
    pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```




```
# calculate D,  d, A, b0
n = length(y)
D = x %*% t(x) * y %*% t(y) + diag(1e-10, n)
d = rep(1, n)
A = t(rbind(t(y), diag(1, n), diag(-1, n)))
C = 1
b0 = c(rep(0, n+1), rep(-C, n))

# solve the quadratic program
alpha = solve.QP(D, d, A, b0, meq = 1)$solution

# thresholding alpha for numerical errors
alpha[alpha < 1e-5] = 0
alpha[alpha > 1 - 1e-5] = 1
```

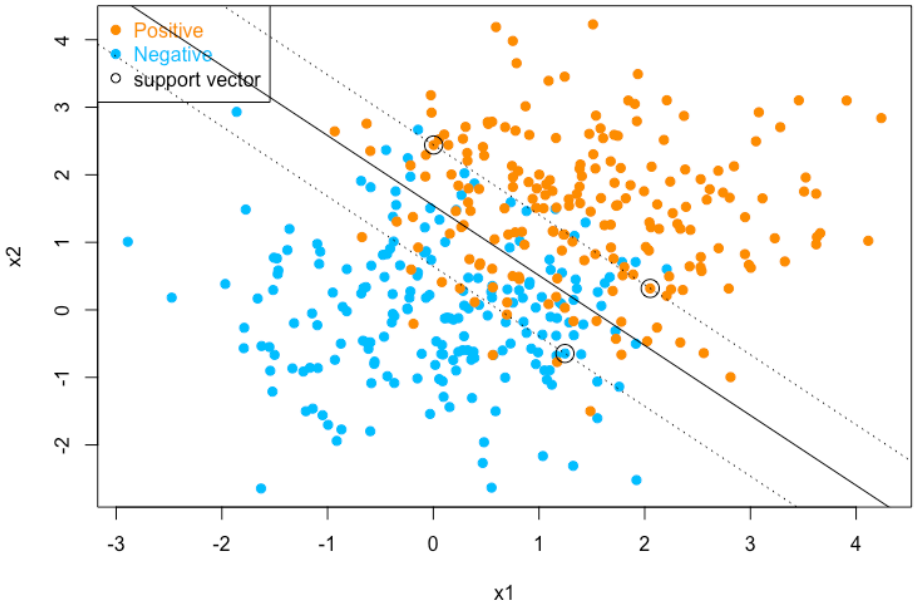
```
# get index for support vectors
support_vec = which(alpha > 1e-2 & alpha < C - 1e-2)

# plot the data
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19,
   xlab = "x1", ylab = "x2")

# plot support vectors
points(x[support_vec, ], pch = 1, cex = 2)
legend("topleft", c("Positive","Negative", "support vector"),
    col=c("darkorange", "deepskyblue", "black"), pch=c(19, 19, 1),
    text.col=c("darkorange", "deepskyblue", "black"))

beta = t(x) %*% (alpha * y)
beta0 = -(mean((x[support_vec, ] %*% beta)[y[support_vec] == -1]) + mean((x[support_vec, ]
%*% beta)[y[support_vec] == 1]))/2

abline(-beta0/beta[2], -beta[1]/beta[2])
abline(-(beta0 - 1)/beta[2], -beta[1]/beta[2], lty=3)
abline(-(beta0 + 1)/beta[2], -beta[1]/beta[2], lty=3)
```




```
# coefficients
print(c(beta0, beta))
## [1] -1.716449  1.151732  1.111536
# training error
train_err = mean((beta0 + x %*% beta) * y < 0)
print(train_err)
## [1] 0.145
```

Question 3 [30 Points] Penalized Loss Linear SVM

We can also perform linear and nonlinear classification using the penalized loss framework. In this question, we will only use the linear version. Use the same dataset in Question 2. Consider the following logistic loss function:

The rest of the job is to solve this optimization problem if given the functional form of . To do this, we will utilize the general-purpose optimization package/function. For example, in R, you can use the optim() function. Read the documentation of this function (or equivalent ones in Python) and set up the objective function properly to solve for the parameters. If you need an example of how to use the optim() function, read the corresponding part in the example file provided on our course website here (Section 10).

We let is to be a linear function, SVM can be solved by optimizing a penalized loss:

You should use the data from Question 2, and answer these questions:

[10 pts] Drive the gradient of this penalized loss function, typeset with LaTex.

Expanding logistic loss in penalized loss function,

Taking derivative with respect to , we get

$$ \frac{\partial L}{\partial\beta_0} = \sum_{i=1}^n \frac {-y_i e^{- y_i (\beta_0 + x_i^T \boldsymbol\beta)}} {1 + e^{- y_i (\beta_0 + x_i^T \boldsymbol\beta)}} \\ = \sum_{i=1}^n \frac {-y_i} {1 + e^{y_i (\beta_0 + x_i^T \boldsymbol\beta)}} $$

Taking derivative with respect to , we get

* [10 pts] Write a penalized loss objective function SVMfn(b, x, y, lambda) and its gradient SVMgn(b, x, y, lambda).

```
# loss function
svmfn <- function(b, x, y, lambda){
 sum(log(1 + exp(- (x %*% b) * y))) + lambda * sum(b[-1]^2)
}

# gradient of loss function
svmgn <- function(b, x, y, lambda){
 e1 = exp(-y*(x %*% b))
 e2 = y*e1 / (1+e1)
 return( - t(x) %*% e2 + c(0, 2*lambda*b[-1]) )
}
```

[10 pts] Solve the coefficients using optim() and your objective and gradient functions with and BFGS method. Use 0 as the initialized value.

```
mysvmfit <- optim(par = rep(0, 3), fn = svmfn, gr = svmgn,
        x = cbind(1, x), y = y,
        lambda = 1, method = "BFGS")
```

Report the followings:

Your coefficients

```
beta0 = mysvmfit$par[1]
beta = mysvmfit$par[2:3]

print(c(beta0, beta))
## [1] -2.280224  1.575649  1.509805
```

Your loss and mis-classification rate on training data.

```
# training err and loss
train_err = 1 - mean( y * (x %*% beta + beta0) > 0)
train_loss = mysvmfit$value
print(c(train_err, train_loss))
## [1]   0.1475 129.8600
```

Plot all data and the decision line

```
plot(x, pch = 19, xlab = "x1", ylab = "x2",
   col = ifelse(y>0,"darkorange", "deepskyblue"))
legend("topleft", c("Positive","Negative"),
     col=c("darkorange", "deepskyblue"),
     pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))

abline(a= -beta0/beta[1], b=-beta[1]/beta[2], col="black", lty=1, lwd = 2)
```