STAT 542: Homework 11

Spring 2021, by Arka Mitra (arkam2)

Due: Monday, Apr 20, 11:59 PM CT

Contents

About HW11 1
Question 1 [50 Points] Sovling SVM using Quadratic Programming 1
Question 2 [50 Points] Penalized Loss SVM 5

About HW11

We practice two approaches for solving SVM: quadratic programming and penalized version. The first one utilize the `quadprog` package, while the second one can be solved using the `optim()` function.

Question 1 [50 Points] Sovling SVM using Quadratic Programming

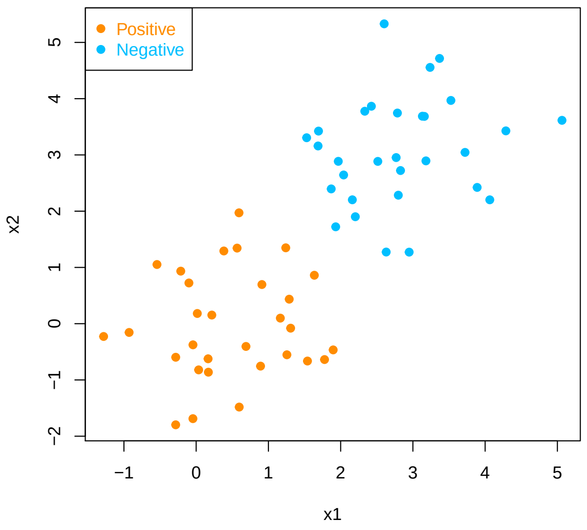
Install the `quadprog` package (there are similar ones in Python too) and utilize the function `solve.QP` to solve SVM. The `solve.QP` function is trying to perform the minimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} b^T \mathbf{D} b - d^T b, \\ \text{subject to} \quad & \mathbf{A}^T b \geq b_0, \end{aligned}$$

where b is the unknown parameter. For more details, read the documentation of the `quadprog` package on CRAN. You should generate the data using the following code (or write a similar code in Python). This is a linearly separable problem.

```r
set.seed(4); n <-30; p <- 2
xpos <- matrix(rnorm(n*p, mean=0, sd=1), n, p)
xneg <- matrix(rnorm(n*p, mean=3, sd=1), n, p)
x <- rbind(xpos, xneg)
y <- matrix(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"))
```



a) [25 points] The Primal Form

Use the formulation defined at page 15 of the `SVM` lecture note. Make sure that you perform the following:

- Let $b = (\beta_0, \boldsymbol{\beta}^T)^T$. Then properly define $\mathbf{D}$, d, $\mathbf{A}$ and b_0 corresponding to this b for our SVM problem.
- Obtain the solution using the `solve.QP` function, and obtain the decision function.
- Plot the decision line, the two margin lines and the support vectors

Note: The package requires $\mathbf{D}$ to be positive definite, while it is not true in our case. To address this problem, add 10^{-5} to the top-left element of your $\mathbf{D}$ matrix, which is the one corresponding to β_0. This will make $\mathbf{D}$ invertible. This may affect your results slightly. So be careful when plotting your support vectors.

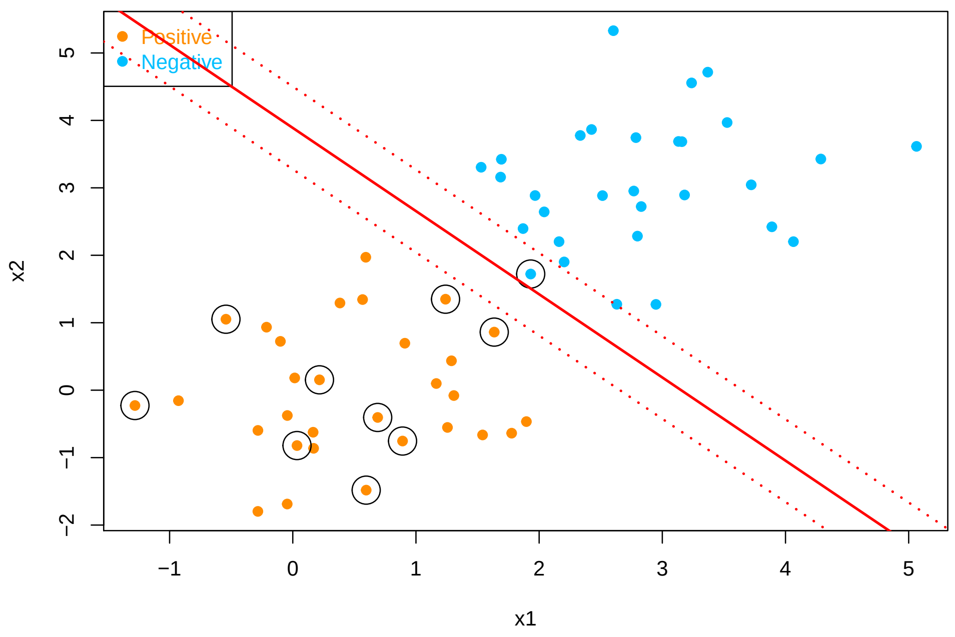
Given, $b = (\beta_0, \boldsymbol{\beta}^T)^T$. Then the matrix A is the product of each row in y and x. Since $\hat{\beta} = Q\beta$, the matrix $Q = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and matrix $D = Q^TQ = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. d is a (p+1)-shaped vector filled with 0 and a is a $n * p$ shaped vector filled with 1.

```r
e2 = 1e-5
Q = sapply(1:(n*p), function(i) y[i]*t(x)[,i])

# matrices
D = t(Q)%*%Q + e2*diag(n*p)
A = t(rbind(matrix(y,1,n*p),diag(n*p)))
# vectors
d_ = matrix(1,n*p,1)
a_ = matrix(0,n*p+1,1)
# dual
dual = solve.QP(D,d_,A,a_,meq=1)
sol = matrix(dual$solution,n*p,1)
# SV, beta & beta0
sv2 = which(abs(sol)>=e2)
b_2 = t(sol) %*% t(Q)
```

```
b0_2 = -sum(x[sv2[-3],]%*%t(b_2))/2

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"))
points(x[sv2, ], col="black", cex=3)
abline(a=-b0_2/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=1, lwd = 2)
abline(a= (-b0_2-1)/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=3, lwd = 2)
abline(a= (-b0_2+1)/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=3, lwd = 2)
```



b) [25 points] The Dual Form

Formulate the SVM **dual** problem in page 24 of the lecture note. Make sure that you perform the following:

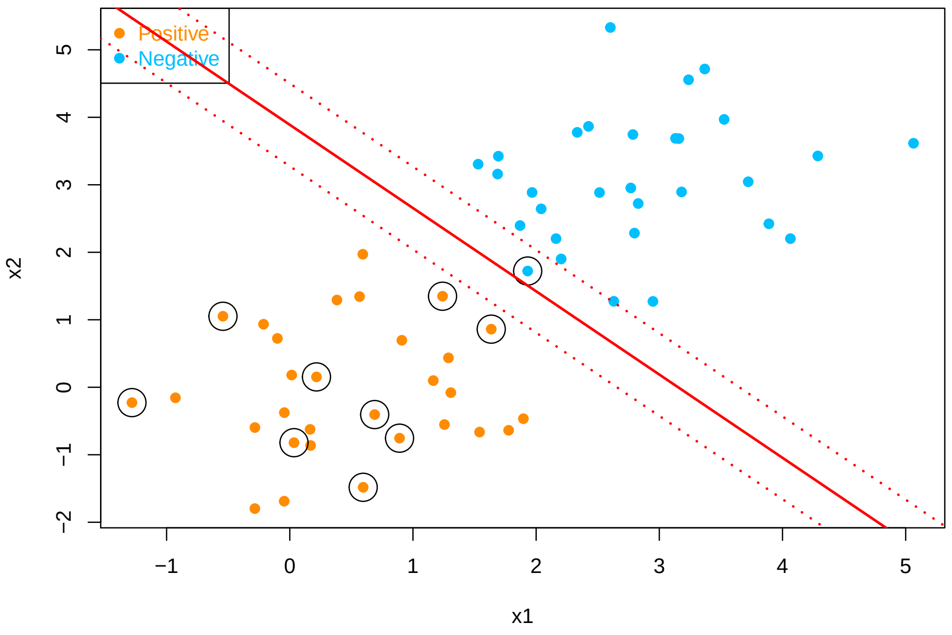
- Let $b = (\alpha_1, \ldots, \alpha_n)^T$. Then properly define $\mathbf{D}$, d, $\mathbf{A}$ and b_0 corresponding to this b for our SVM problem.
- Equality constrains can be addressed using the `meq` argument.
- Obtain the solution using the `solve.QP` function, and convert the solution into $\boldsymbol{\beta}$ and β_0.
- Plot the decision line, the two margin lines and the support vectors

Note: Again, $\mathbf{D}$ is may not be positive definite. This time, add 10^{-6} to all diagonal elements to $\mathbf{D}$. This may affect your results slightly. So be careful when plotting your support vectors.

Given,$b = (\alpha_1, \ldots, \alpha_n)^T$. Thus, matrix Q is the product of each row in y and x while $D = Q^T Q$. Then A is defined as a $(n * p) \times (n * p + 1)$ matrix with the 1st column to be y and the rest of the matrix is diagonal. d is a vector of 1 in length of $n * p$ and a is a vector of 0 in length of $n * p + 1$.

```r
e2 = 1e-5
Q = sapply(1:(n*p), function(i) y[i]*t(x)[,i])
# matrices
D = t(Q)%*%Q + e2*diag(n*p)
A = t(rbind(matrix(y,1,n*p),diag(n*p)))
# vectors
d_ = matrix(1,n*p,1)
a_ = matrix(0,n*p+1,1)
# dual
dual = solve.QP(D,d_,A,a_,meq=1)
sol = matrix(dual$solution,n*p,1)
# SV, beta & beta0
sv2 = which(abs(sol)>=e2)
b_2 = t(sol) %*% t(Q)
b0_2 = -sum(x[sv2[-3],]%*%t(b_2))/2

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"))
points(x[sv2, ], col="black", cex=3)
abline(a=-b0_2/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=1, lwd = 2)
abline(a= (-b0_2-1)/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=3, lwd = 2)
abline(a= (-b0_2+1)/b_2[2], b=-b_2[1]/b_2[2], col="red", lty=3, lwd = 2)
```



Question 2 [50 Points] Penalized Loss SVM

We can also perform linear and nonlinear classification using the penalized loss framework. Consider the following logistic loss function:

$$L(y, f(x)) = \log(1 + e^{-yf(x)}).$$

The rest of the job is to solve this optimization problem if given the functional form of $f(x)$. 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).

Question a) [25 Points] Linear SVM

When $f(x)$ is a linear function, SVM can be solved by optimizing the penalized loss:

$$\underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \sum_{i=1}^{n} L(y_i, \beta_0 + x_i^T \boldsymbol{\beta}) + \lambda \|\beta\|^2$$

You should use the data from Question 1, and answer these questions:

- Write a function to define the penalized loss objective function. The R function `optim()` can run faster if you further define the gradient function. Hence, you should also define the gradient function properly and implement it in the optimization.
- Choose a reasonable λ value so that your optimization can run properly. In addition, I recommend using the `BFGS` method in optimization.
- After solving the optimization problem, plot all data and the decision line
- If needed, modify your λ so that the model fits reasonably well and re-plot. You don't have to tune λ as long as you obtain a reasonable decision line.
- Report your training error.

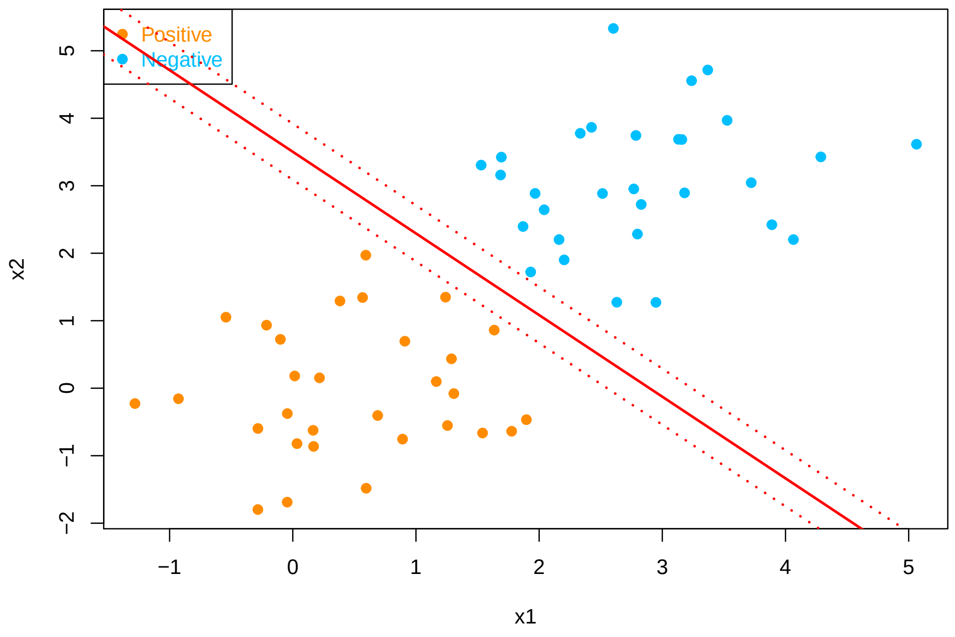
```r
x1 = cbind(matrix(1,n*p,1),x)
# loss function
loss <- function(b){
  b = matrix(b)
  t1 = sum(log(1+exp(-y*(x1%*%b))))
  t2 = 0.1*t(b[-1,])%*%b[-1,]
  return(t1+t2)
}
# gradient function
grad <- function(b){
  b = matrix(b)
  g1 = -1/(1+exp(-y*(x1%*%b)))*exp(-y*(x1%*%b))
  g2 = sapply(1:(n*p),function(i) y[i]*matrix(x1[i,]))
  g3 = apply(sapply(1:(n*p),function(i) g2[,i]*g1[i]),1,sum)
  g22 = 2*0.1*matrix(c(0,b[-1,]))
  return(matrix(g3)+g22)
}
# optimization
opt1 = optim(rep(0,3),fn=loss,gr=grad,method="BFGS")
beta0_1 = opt1$par[1]
beta_1 = opt1$par[-1]
beta_1
```

```
## [1] -2.910629 -2.406226
```

```
beta0_1
```

```
## [1] 8.429286
```

```
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"))
abline(a=-beta0_1/beta_1[2], b=-beta_1[1]/beta_1[2], col="red", lty=1, lwd = 2)
abline(a= (-beta0_1-1)/beta_1[2], b=-beta_1[1]/beta_1[2], col="red", lty=3, lwd = 2)
abline(a= (-beta0_1+1)/beta_1[2], b=-beta_1[1]/beta_1[2], col="red", lty=3, lwd = 2)
```



Question b) [25 Points] Non-linear SVM

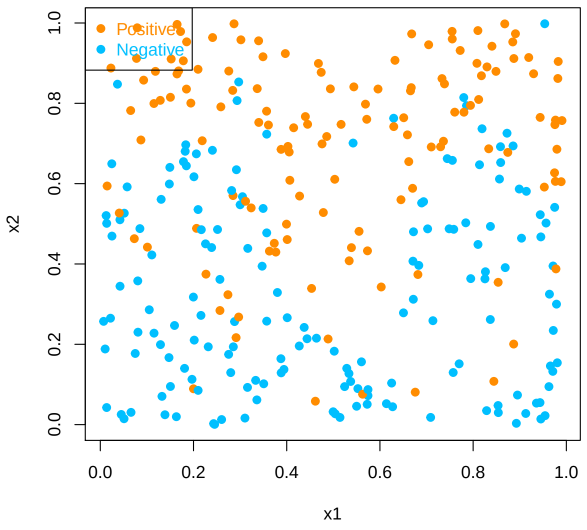
You should generate the data using the code provided below (or write similar code in Python):

```
  set.seed(2)
  n = 300
  p = 2 # dimension

  # Generate the positive and negative examples
  x <- matrix(runif(n*p), n, p)
  side <- (x[, 2] > 0.5 + 0.3*sin(3*pi*x[, 1]))
  y <- sample(c(1, -1), n, TRUE, c(0.9, 0.1))*(side == 1) + sample(c(1, -1), n, TRUE, c(0.1, 0.9))*(sid

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

```r
legend("topleft", c("Positive","Negative"),
       col=c("darkorange", "deepskyblue"), pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```



We will utilize the Reproducing Kernel Hilbert Space (RKHS) to search for our decision function $f(x)$. This means that the decision function can be represented using

$$\widehat{f}(x) = \sum_i w_i K(x, x_i)$$

If we plug this into our loss function, it becomes

$$\sum_{i=1}^{n} L(y_i, \mathbf{w}^T K_i) + \lambda \mathbf{w}^T \mathbf{K} \mathbf{w}$$

where $\mathbf{K}$ is an $n \times n$ kernel matrix, with the (i, j)th entry being K is $K(x_i, x_j)$, K_i is the ith column of $\mathbf{K}$, and $\mathbf{w} = (w_1, \ldots, w_n)$. For this problem, let's consider the Gaussian kernel $K(\cdot, \cdot)$, which we have used in the kernel regression lecture. For the bandwidth of the kernel function, you can borrow ideas from our previous homework. But you do not need to optimize the bandwidth. Perform the following steps to complete this model fitting:

- Pre-calculate the kernel matrix $\mathbf{K}$. Fix a σ^2 value (in the Gaussian density function), as long as your optimization runs properly.
- Write a function to define the objective function. You should also define the gradient function properly and implement it in the optimization. Type your formula of the gradient function in Latex.
- It could be difficult to obtain the decision line itself. However, its relatively easy to obtain the fitted label. Hence, calculate the fitted label (in-sample prediction) of your model and report the classification error.
- Plot the data using the fitted labels, and add the true decision line to the plot. This would allow you to visualize (approximately) the decision line. If needed, modify your λ so that the model fits reasonably well and re-plot. You don't have to tune λ as long as your result is reasonable.

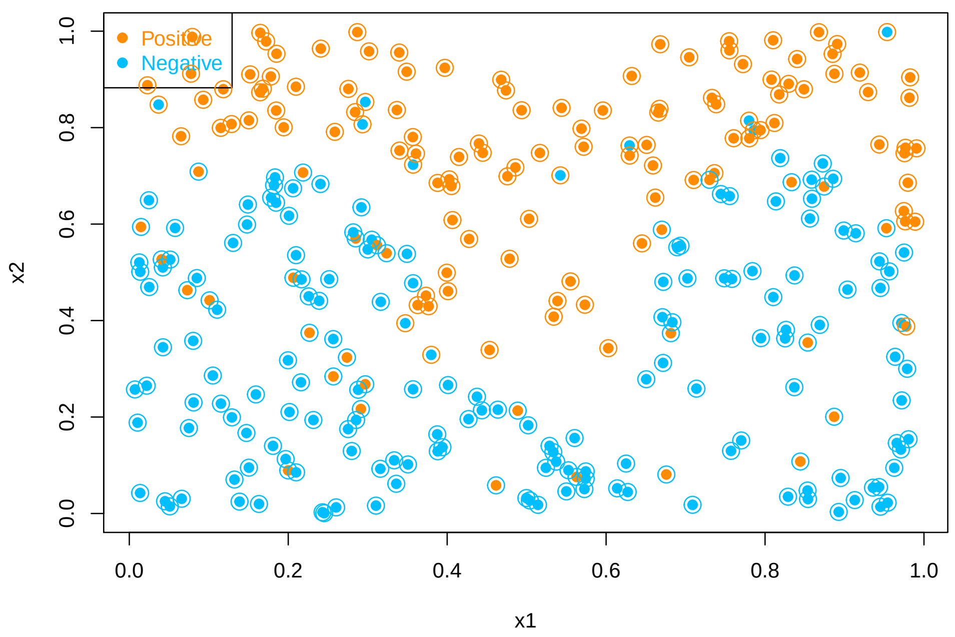
```r
# Gaussian Kernel
KGauss <- function(u){
  1 / sqrt(2*pi) * exp(-(u^2)/2)
}
lambda = apply(x,2,sd)*(4/3/nrow(x))^(1/5)
Kfunc <- function(u) {KGauss(u/lambda)/lambda}
K = matrix(NA,n,n)

for (i in 1:n){
  K[,i] = apply(sapply(1:n, function(j) Kfunc(x[i,]-x[j,])),2,mean)
}

# loss function
loss <- function(b){
  b = matrix(b)
  t1 = sum(log(1+exp(-y*(K%*%b))))
  t2 = 0.5*t(b[-1,])%*%b[-1,]
  return(t1+t2)
}
# gradient function
grad <- function(b){
  b = matrix(b)
  g1 = -1/(1+exp(-y*(K%*%b)))*exp(-y*(K%*%b))
  g2 = sapply(1:n,function(i) y[i]*matrix(K[i,]))
  g3 = apply(sapply(1:n,function(i) g2[,i]*g1[i]),1,sum)
  g22 = 2*0.5*matrix(c(0,b[-1,]))
  return(matrix(g3)+g22)
}
# optimization
opt2 = optim(rep(0,n),fn=loss,gr=grad,method="BFGS")
beta_2 = opt2$par
pred_y = sign(K%*%beta_2)
print(paste0("The misclassification rate is: ", mean(pred_y!=y)))
```

```
## [1] "The misclassification rate is: 0.133333333333333"
```

```r
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), data = pred_y[,1], pch = 19, xlab = "x1", ylab = "x
legend("topleft", c("Positive","Negative"),
       col=c("darkorange", "deepskyblue"), pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
points(x, col=ifelse(pred_y>0,"darkorange", "deepskyblue"), cex=1.8)
```


Positive
Negative
x1
x2
0.0
0.2
0.4
0.6
0.8
1.0