STAT 542 / CS 598: Homework 6

Fall 2019, by Regan Chan (ttchan2)

Due: Monday, Nov 11 by 11:59 PM Pacific Time

Contents

Question 1 [50 Points] Linearly Separable SVM using Quadratic Programming

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

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

For more details, read the document file of the `quadprog` package on CRAN. Investigate the dual optimization problem of the seperable SVM formulation, and write the problem into the above form by properly defining $\mathbf{D}$, d, A and a.

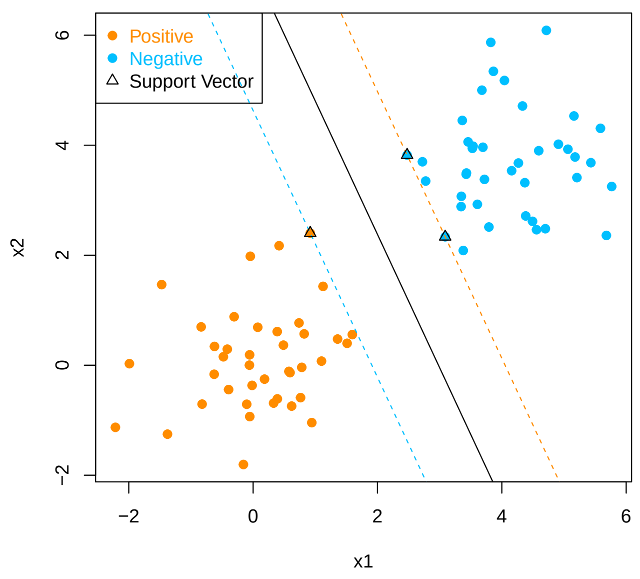
Note: The package requires $\mathbf{D}$ to be positive definite, while it may not be true in our problem. A workaround is to add a "ridge," e.g., $10^{-5}\mathbf{I}$, to the $\mathbf{D}$ matrix, making it invertible. This may affect your later results, so figure out a way to fix them.

You should generate the data using the following code (or write a similar code in Python). After solving the quadratic programming problem, perform the following:

- Convert the solution into β and β_0, which can be used to define the classification rule
- Plot all data and the decision line
- Add the two separation margin lines to the plot
- Add the support vectors to the plot

```r
set.seed(1); n <-40; p <- 2
xpos <- matrix(rnorm(n*p, mean=0, sd=1), n, p)
xneg <- matrix(rnorm(n*p, mean=4, sd=1), n, p)
x <- rbind(xpos, xneg)
y <- matrix(c(rep(1, n), rep(-1, n)))

library(quadprog)
xy <- x * y[,1]
D <- xy %*% t(xy)
diag(D) <- diag(D) + 1e-5
d <- rep(1, 2*n)
A <- rbind(y[,1], diag(2*n))
bvec <- rbind(rep(0, 2*n+1))
soln <- solve.QP(D,d,t(A),bvec=bvec, meq=1)

b <- colSums((soln$solution * y)[,1] * x)
neg_x <- x[y==-1,]
pos_x <- x[y==1,]
b0 <- -mean(c(max(neg_x %*% b), min(pos_x %*% b)))
```

```r
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", ylab = "x2")
legend("topleft", c("Positive","Negative","Support Vector"),
     col=c("darkorange", "deepskyblue","black"), pch=c(19, 19, 2), text.col=c("darkorange", "deepskyblu

decisionLine <- function(b, b0, ...) {
  abline(-b0/b[2], -b[1]/b[2], ...)
}

decisionLine(b, b0)
decisionLine(b, b0+1, col="darkorange", lty=2)
decisionLine(b, b0-1, col="deepskyblue", lty=2)

supportVectors <- x[soln$solution > 10e-5,]
points(supportVectors[,1], supportVectors[,2], pch=2)
```



Question 2 [25 Points] Linearly Non-seperable SVM using Penalized Loss

We also introduced an alternative method to solve SVM. Consider a logistic loss function

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

and solve the penalized loss for a linear SVM

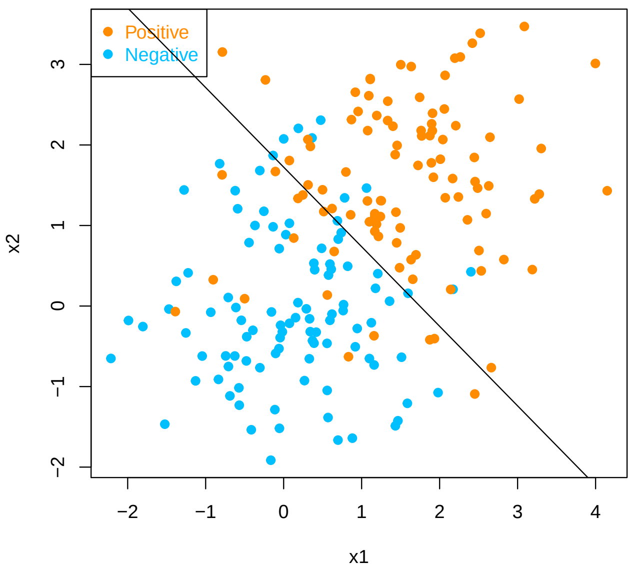
$$\underset{\beta_0, \beta}{\arg\min} \sum_{i=1}^{n} L(y_i, \beta_0 + x^T\beta) + \lambda\|\beta\|^2$$

The rest of the job is to solve this optimization problem. To do this, we will utilize a 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 provide on our course website here (Section 10). You should generate the data using the following code (or write a similar code in Python). Perform the following:

- Write a function to define the objective function (penalized loss). The algorithm may run faster if you further define the gradient function. However, the gradient is not required for completing this homework, but it counts for 2 bonus points.

- Choose a reasonable λ value so that your optimization can run properly. In addition, I recommend using the BFGS method in the 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 (you do not have to optimize this tuning), and re-plot

```r
set.seed(1)
n = 100 # 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))

lambda <- 1
L <- function(param) {    # param = c(b0, b)
  sum(log(1+exp(-y * (param[1] + x %*% param[2:3])))) + lambda*sum(param[2:3]^2)
}
dL <- function(param) {    # param = c(b0, b)
  e <<- (exp(y*(param[1] + x %*% param[2:3]))+1)[,1]
  g <<- c(
    sum(-y/e),
    colSums((2*lambda*param[2:3]*e - x*y) / e)
  )
  return(g)
}

soln <- optim(c(0, -1, 0), L, dL, method="BFGS")

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"))
b0 <- soln$par[1]
b <- soln$par[2:3]
decisionLine(b, b0)
```



Question 3 [25 Points] Nonlinear and Non-seperable SVM using Penalized Loss

We can further use the kernel trick to solve for a nonlinear decision rule. The optimization becomes

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

where K_i is the ith column of the $n \times n$ kernel matrix K. For this problem, we consider the Gaussian kernel (you do not need an intercept). Again, we can use the logistic loss.

You should generate the data using the following code (or write a similar code in Python). Perform the following:

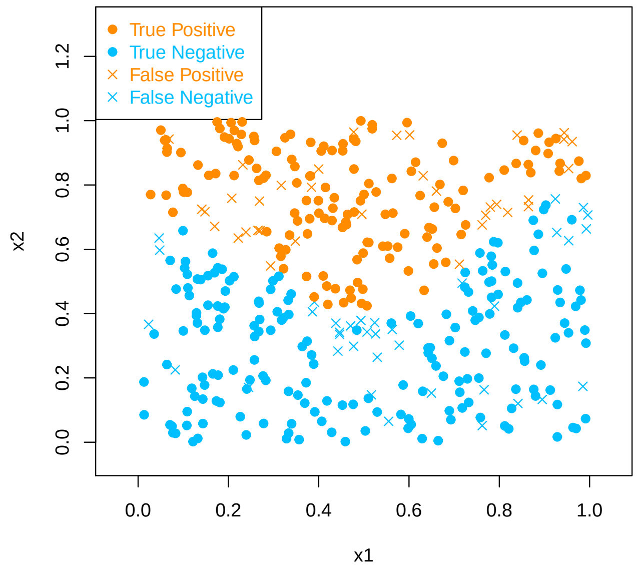
- Pre-calculate the $n \times n$ kernel matrix K of the observed data
- Write a function to define the objective function (this should not involve the original x, but uses K). Again, the gradient is not required for completing this homework. However, it counts for 3 bonus points.
- Choose a reasonable λ value so that your optimization can run properly
- After solving the optimization problem, plot **fitted** labels (in-sample prediction) for all subjects
- If needed, modify your λ so that the model fits reasonably well (you do not have to optimize this tuning), and re-plot
- Summarize your in-sample classification error

Ans: With gaussian kernel, the accuracy is about 80%

```
  set.seed(1)
  n = 400
  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

#  library(KRLS)
  lambda <- 1
  myGaussKernel <- function(m, sigma) {
    m <- t(m)
    exp(-sigma*apply(m, 2, function(col){
      colSums((m-col)^2)
    }))
  }
  K <- myGaussKernel(x, 5)
  L <- function(b) {
    sum(log(exp(-y*(K %*% b))+1)) + lambda*(b %*% K %*% b)[1,1]
  }
  dL <- function(b) {
    colSums(-K*y/((exp(y*(K %*% b))+1))[,1]) + 2*lambda*K %*% b
  }

  soln <- optim(rep(1, n), L, dL, method="BFGS")
  pred <- soln$par %*% K
  pred <- pred / abs(pred)
  paste("accuracy:", sum(pred == y)/length(y))
```

```
## [1] "accuracy: 0.8225"
```

```
  plot(0, type="n", pch = 19, xlab = "x1", ylab = "x2", xlim=c(-.05, 1.05), ylim=c(-.05, 1.3))
  legend("topleft",
    c("True Positive","True Negative", "False Positive", "False Negative"),
    col=c("darkorange", "deepskyblue", "darkorange", "deepskyblue"),
    pch=c(19, 19, 4, 4),
    text.col=c("darkorange", "deepskyblue", "darkorange", "deepskyblue"),
  )

  points(x[,1], x[,2], col=ifelse(pred>0, "darkorange", "deepskyblue"), pch=ifelse(pred==y, 19, 4))
```


● True Positive
● True Negative
× False Positive
× False Negative
x1
x2