STAT 542 / CS 598: Homework 6

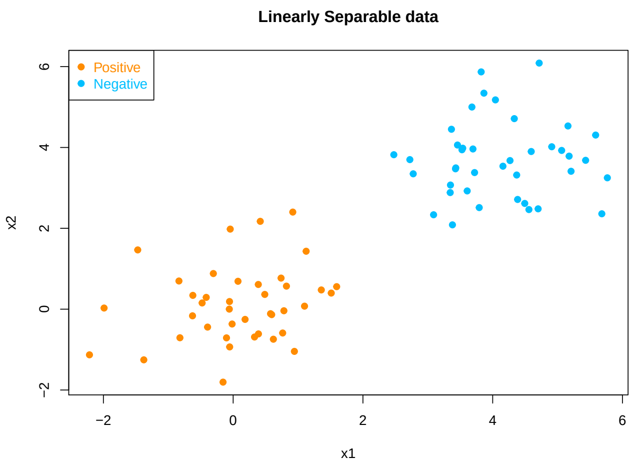
Pushpit Saxena (netid: pushpit2)

Contents

Question 1 Linearly Separable SVM using Quadratic Programming

- Data Generation

```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 <- as.matrix(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",
     main="Linearly Separable data")
legend("topleft", c("Positive","Negative"),
     col=c("darkorange", "deepskyblue"),
     pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```



Solving dual SVM using quadprog

- Generated the appropriate parameters for **solve.QP** function.
- Converted the solution into β and β_0

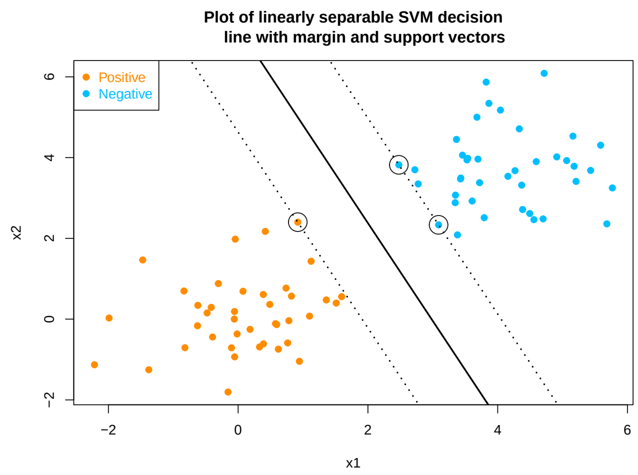
```
set.seed(1)
library(quadprog)
n <- dim(x)[1]
eps <- 10e-5

# Method to convert the alpha values obtained from solve.QP function
# to beta (denoted by 'W') and beta_0 (denoted by 'b'). Also used those
# values to generate intercept and slope for decision-line, and two
# margin lines
findLine <- function(a, y, X) {
  nonzero <-  abs(a) > 1e-5
  W <- rowSums(sapply(which(nonzero), function(i) a[i]*y[i]*X[i,]))
  b <- -(max(X[y == -1, ] %*% W) + min(X[y == 1, ] %*% W))/2
  slope <- -W[1]/W[2]
  intercept <- -b/W[2]
  intercept_1 <- (-b-1)/W[2]
  intercept_2 <- (-b+1)/W[2]
  return(c(intercept,slope, intercept_1, intercept_2))
}

# Generating appropriate parameters for solve.QP function to solve Dual SVM
Q <- sapply(1:n, function(i) y[i]*t(x)[,i])
D <- t(Q)%*%Q
d <- matrix(1, nrow=n)
b0 <- rbind( matrix(0, nrow=1, ncol=1) , matrix(0, nrow=n, ncol=1) )
A <- t(rbind(matrix(y, nrow=1, ncol=n), diag(nrow=n)))

# call the QP solver:
sol <- solve.QP(D +eps*diag(n), d, A, b0, meq=2, factorized=FALSE)
qpsol <- matrix(sol$solution, nrow=n)
```

Plotting the results

- Plotted all data and the decision line

- Added the two separation margin lines to the plot
- Added the support vectors to the plot - **Support vectors are encircled in the plot**

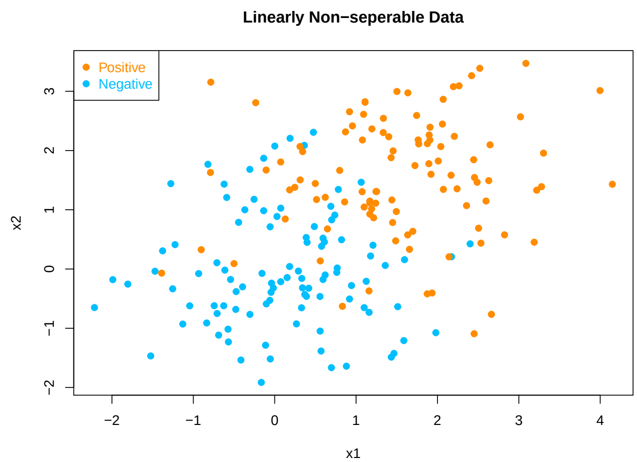
```
qpline <- findLine( qpsol, y, x)
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"),
     pch = 19, xlab = "x1", ylab = "x2",
     main="Plot of linearly separable SVM decision
     line with margin and support vectors")
legend("topleft", c("Positive","Negative"),
       col=c("darkorange", "deepskyblue"),
       pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
abline(a= qpline[1], b=qpline[2], col="black", lty=1, lwd = 2)
points(x[which(qpsol > 1e-5), ], col="black", cex=3)
abline(a= qpline[3], b=qpline[2], col="black", lty=3, lwd = 2)
abline(a= qpline[4], b=qpline[2], col="black", lty=3, lwd = 2)
```



Question 2 Linearly Non-seperable SVM using Penalized Loss

- Data Generation:

```
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))

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



Function to define the objective function (penalized loss).

- Implemented based on the given penalized logistic loss:

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

```
penalized.loss <- function(b, X, Y, lamda=1e-5) {
      sum(log(1 + exp(-1 * (Y * (b[1] + X%*%b[-c(1)]))))) + (lamda * sum(b^2))
    }
```

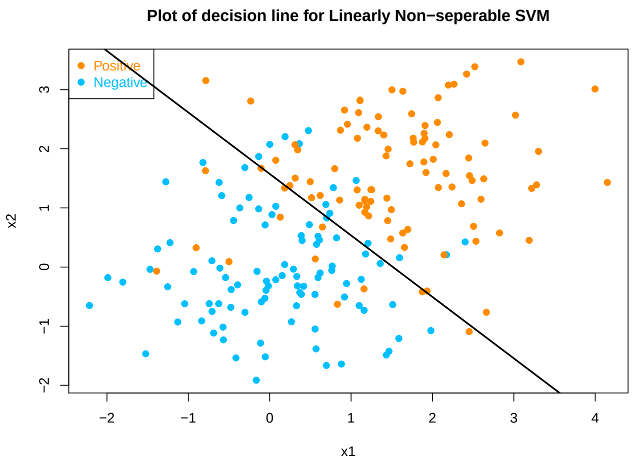
Choosen a reasonable λ (=1e-5) value

```
linear.nonsep.svm <- function(x, y, lamda) {
    b <- rep(0.1, 3)
    sln <- optim(b, penalized.loss, X=x, Y=y, lamda=lamda, method = "BFGS")
    return (sln)
}
```

```r
lamda <- 1e-5
sln <- linear.nonsep.svm(x, y, lamda)
```

Plot of all data and the decision line

```r
    plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"),
         pch = 19, xlab = "x1", ylab = "x2",
         main="Plot of decision line for Linearly Non-seperable SVM")
  legend("topleft", c("Positive","Negative"), col=c("darkorange", "deepskyblue"),
         pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
  abline(a= -sln$par[1]/sln$par[3],
         b=-sln$par[2]/sln$par[3], col="black", lty=1, lwd = 2)
```



If needed, modify your λ so that the model fits reasonably well (you do not have to optimize this tuning), and re-plot

- Peformed a grid search on some values of λ and re-fit and plot the decision boundary again
- I have collected the InSampleFitAccuracy for all the values of λ which can be seen by printing the **results** data frame (*not included in the pdf report*)

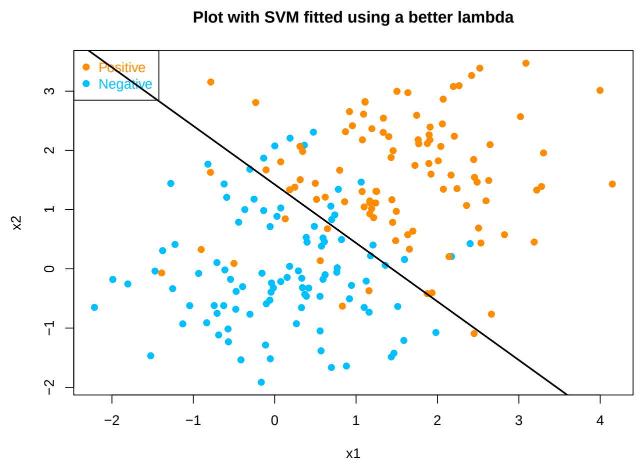
```r
lamdas <- c(1e-6, 1e-5, 1e-4, 1e-3, 1e-2,
            0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 1, 10, 100)
results <- data.frame("Lambda" = c(0), "FitAccuracy" = c(0))

lambdaWithBestAcc <- lamdas[1]
bestAccuracy <- 0
for (lamda in lamdas) {
  sln <- linear.nonsep.svm(x, y, lamda)
  pred_y <- ifelse(x%*% sln$par[-c(1)] + sln$par[1] > 0, 1, -1)
  acc <- length(which(y == pred_y))/length(pred_y)
  results <- rbind(results, c(lamda, acc))
  if(acc > bestAccuracy) {
    lambdaWithBestAcc <- lamda
    bestAccuracy <- acc
  }
}
```

- Best Accuracy (for λ):

```
## Achieved best accuracy for [ Lambda = 1 ]
```

- Plot of the data with decision line calculated using best λ (=1) found above.

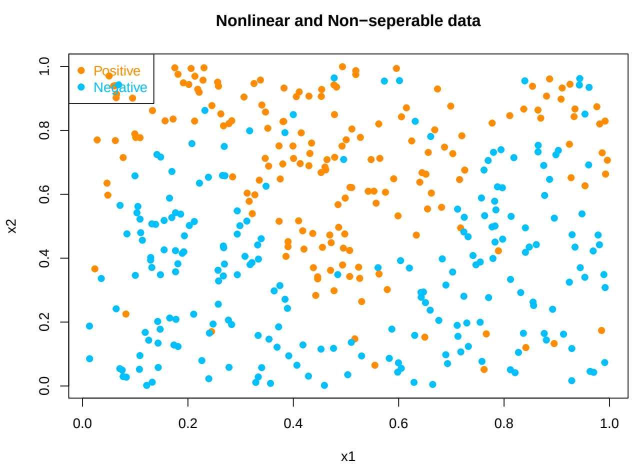
```r
sln <- linear.nonsep.svm(x, y, lambdaWithBestAcc)
    plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"),
         pch = 19, xlab = "x1", ylab = "x2",
         main="Plot with SVM fitted using a better lambda")
  legend("topleft", c("Positive","Negative"),
         col=c("darkorange", "deepskyblue"),
         pch=c(19, 19),
         text.col=c("darkorange", "deepskyblue"))
  abline(a= -sln$par[1]/sln$par[3],
         b=-sln$par[2]/sln$par[3], col="black", lty=1, lwd = 2)
```



Question 3 Nonlinear and Non-seperable SVM using Penalized Loss

- Data Generation

```
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))*(side == 0)

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



Pre-calculate the $n \times n$ kernel matrix K of the observed data.

- I have followed **this** to generate the kernel matrix, using the following formula

$$K(x, x') = \exp(-\gamma \|x - x'||^2)$$

where

$$\gamma = 1/2\sigma^2$$

```
gauss.kernel <- function(X, gamma=1) {
  exp(-gamma * (as.matrix(dist(X))^2))
}
```

Objective function based on the Gaussian Kernel defined earlier

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

```r
rbf.penalized.loss <- function(b, X, Y, lamda=1e-4) {
  sum(log(1 + exp(-1 * (Y * (X%*%b))))) + (lamda * (t(b)%*%X%*%b))
}
```

Choose a reasonable λ value so that your optimization can run properly

- I have chosen a **gamma** ($=1/2\sigma^2$) value to be used in gaussian kernel to be **10** (picked based on the param tuning shown later). I also found out the as we increase this value to a higher value we can achieve 100% accuracy on training data, which clearly leads to over-fitting (shown and explained later), hence picked this value as a trade-off.
- For λ I have picked **1e-5** (also based on the param tuning) giving a reasonably good fit.

```r
rbf.svm <- function(x, y, gamma=10, lamda=1e-5) {
  # gamma <- 10
  K <- gauss.kernel(X=x, gamma=gamma)
  b <- rep(0.1, dim(K)[2])
  # lamda <- 1e-5
  sln <- optim(b, rbf.penalized.loss, X=K, Y=y, lamda=lamda, method = "BFGS")
  d <- sln$par %*% K
  # t(d)
  pred_y <- rep(1, length(y))
  pred_y[which(t(d) < 0)] <- -1
  return (pred_y)
}
pred_y <- rbf.svm(x, y)
```

- Accuracy of the prediction:

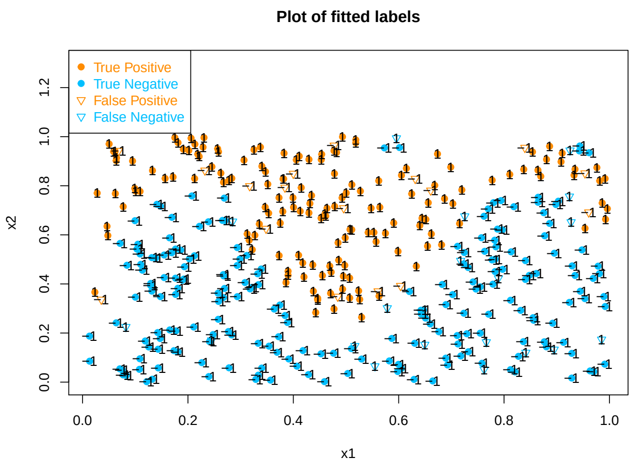
```r
length(which(y == pred_y))/length(pred_y)
```

```
## [1] 0.91
```

Plot fitted labels (in-sample prediction) for all subjects

- The colors in the plot are based on the predicted labels.
- I have also added the true label for each data point as text
- Changed the shape of the observation to triangle when it is a mis-classification

```r
plot.rbf.svm <- function(x, y, pred_y) {
  dat <- data.frame(x= x[,1], y = x[,2])
  dd <- pred_y == y
  plot(y ~ x, data=dat, col=ifelse(pred_y>0,"darkorange", "deepskyblue"),
       pch = ifelse(dd, 19, 25),
       xlab = "x1", ylab = "x2", ylim=c(0,1.3), main="Plot of fitted labels")
  legend("topleft", c("True Positive","True Negative",
                      "False Positive", "False Negative"),
         col=c("darkorange", "deepskyblue", "darkorange", "deepskyblue"),
         pch=c(19, 19, 25, 25),
         text.col=c("darkorange", "deepskyblue", "darkorange", "deepskyblue"))
  text(dat$x, dat$y, label=ifelse(y > 0, "1", "-1"))
}
plot.rbf.svm(x,y,pred_y)
```



If needed, modify your λ so that the model fits reasonably well (you do not have to optimize this tuning), and re-plot

- Below is the code for tuning hyperparamenters (γ and λ)
- I have already picked a reasonably well performing set of hyperparameters (*gamma = 10 and lamda = 1e-5*) which provides an accuracy of 0.91, the plot can be seen above, so not sure what exactly to re-plot here (but plotted the fitted labels based on the SVM model with *gamma=50 and lambda=1e-5*, which gives slightly better accuracy). Please note that if we keep on increasing γ or keep on decreasing λ we can achieve 100% accuracy on training sample due to overfitting (can be seen from the accuracy data later, I have also tried to explain the reasoning behind this in that section)

```
gammas <- c(0.001, 0.1, 0.5, 0.9, 1, 10, 50, 100)
lamdas <- c(1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5)
results <- data.frame("Iteration" = c(0), "Gamma" = c(0), "Lambda" = c(0), "Accuracy" = c(0), strings
iter <- 1
start.time <- Sys.time()
for (gamma in gammas) {
  for (lamda in lamdas) {
      K <- gauss.kernel(X=x, gamma=gamma)
      b <- rep(0.1, dim(K)[2])
      sln <- optim(b, rbf.penalized.loss, X=K, Y=y, lamda=lamda, method = "BFGS")
      d <- sln$par %*% K
      pred_y <- rep(1, length(y))
      pred_y[which(t(d) < 0)] <- -1
      acc <- length(which(y == pred_y))/length(pred_y)
      results <- rbind(results, c(iter, gamma, lamda, acc))
      iter <- iter + 1
  }
}
# Sys.time() - start.time
```

- Replotting (with hyper-parameters providing better in sample accuracy, as mentioned above):

```
pred_y <- rbf.svm(x,y,50,1e-5)
```

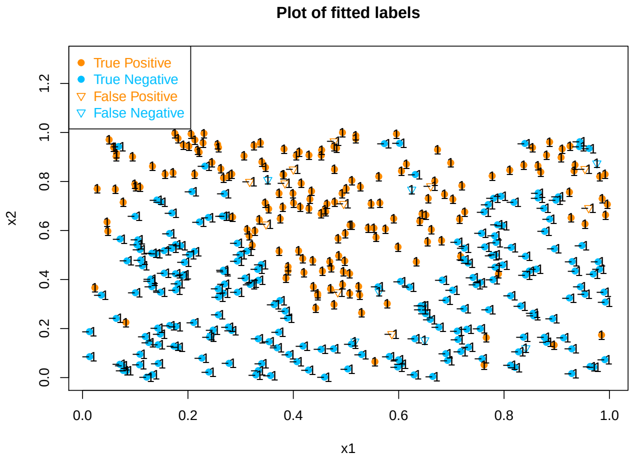
```
+ Accuracy of the prediction:
```

```
length(which(y == pred_y))/length(pred_y)
```

```
## [1] 0.9525
```

+ Plot:

```
plot.rbf.svm(x, y, pred_y)
```



Summarize your in-sample classification error

The behavior of the model is very sensitive to the γ parameter. If γ is too large, the support vector's radius of the area of influence only includes the support vector itself leading to over-fitting and when the γ is very small, the model is too constrained (which leads to underfitting). Similarly large values of λ increases the regularization and leads to underfitting vs smaller values of λ.

- Please find below the accuracy of the in sample predictions at different values of γ and λ

```
rr <- results[-1,]
rownames(rr) <- NULL
rr[, -c(1)]
```

Gamma	Lambda	Accuracy
1e-03	1e+00	0.5550
1e-03	1e-01	0.5550
1e-03	1e-02	0.7450
1e-03	1e-03	0.7700
1e-03	1e-04	0.7700
1e-03	1e-05	0.7700
1e-01	1e+00	0.7450
1e-01	1e-01	0.7750
1e-01	1e-02	0.7700
1e-01	1e-03	0.7825
1e-01	1e-04	0.7850
1e-01	1e-05	0.7850
5e-01	1e+00	0.7725
5e-01	1e-01	0.7750
5e-01	1e-02	0.7925
5e-01	1e-03	0.8075
5e-01	1e-04	0.8250
5e-01	1e-05	0.8275
9e-01	1e+00	0.7700
9e-01	1e-01	0.7825
9e-01	1e-02	0.8100
9e-01	1e-03	0.8250

Gamma	Lambda	Accuracy
9e-01	1e-04	0.8775
9e-01	1e-05	0.8775
1e+00	1e+00	0.7675
1e+00	1e-01	0.7850
1e+00	1e-02	0.8100
1e+00	1e-03	0.8350
1e+00	1e-04	0.8700
1e+00	1e-05	0.8800
1e+01	1e+00	0.8625
1e+01	1e-01	0.8850
1e+01	1e-02	0.8975
1e+01	1e-03	0.9100
1e+01	1e-04	0.9075
1e+01	1e-05	0.9100
5e+01	1e+00	0.8950
5e+01	1e-01	0.9175
5e+01	1e-02	0.9275
5e+01	1e-03	0.9325
5e+01	1e-04	0.9375
5e+01	1e-05	0.9525
1e+02	1e+00	0.9100
1e+02	1e-01	0.9275
1e+02	1e-02	0.9400
1e+02	1e-03	0.9750
1e+02	1e-04	0.9950
1e+02	1e-05	1.0000

Conclusion:

The RBF kernel based penalized log loss SVM is sensitive to the γ parameter of the kernel as well as regularization constant λ. I have picked the values for both that are performing reasonably good without overfitting. A much better approach might be to create a holdout (validation) set and use it to pick the best values for these parameters (which is out of the scope of this excersize).