# STAT542 HW3

**Name: Chun Yin Ricky Chue (chue2@illinois.edu)**

## Question 1.

### Part a.

We generate a set of separable data.

```
library(quadprog); library("e1071"); 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)))
```

Write the equation of the separating hyperplane, $\mathbf{w}^T\mathbf{x} + b = 0$. In the separable case, we want the support vectors to be on the margins. Hence, the objective function that is to be minimized, $g : \frac{1}{2}||\mathbf{w}||^2$, and subject to the constraint is $h : y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$, for $i = 1, 2, ..., N$, where $N = 2n$, the total number of data points.

From the minimization problem solved by `solve.QP`, we see that we are using 2 features, $X_1$ and $X_2$ to predict the response $Y$, so number of features $p = 2$. Also, $\mathbf{w}$ is $p \times 1 = 2 \times 1$, `i.e.` $\mathbf{w} = (w_1, w_2)^T$, and $Y$ is $N \times 1$. We can write formulate the problem as follows:

$$\beta^T = \begin{bmatrix} w_1, w_2, b \end{bmatrix} \quad , \quad \mathbf{D} = \begin{bmatrix} I_{p\times p} & 0_{p\times 1} \\ 0_{1\times p} & 0_{1\times 1} \end{bmatrix} \quad , \quad b_0^T = \begin{bmatrix} 1, 1, \cdots, 1 \end{bmatrix}_{1\times N} \quad , \quad d^T = \begin{bmatrix} 0, 0, 0 \end{bmatrix}$$

Hence, $\beta$ and $d$ are $(p+1) \times 1$ vectors, $\mathbf{D}$ is a $(p+1) \times (p+1)$ matrix, and $b_0$ is a $N \times 1$ vector.

Comparing the constraints of this Primal problem and the one in `solve.QP`, one can see that $(\mathbf{A}^T\beta)_i = y_i(\mathbf{w} \cdot \mathbf{x_i} + b)$, for $i = 1, 2, ..., N$. Comparing the dimensions of $\mathbf{A}^T$, $\beta$ and $b_0$, for the matrix calculation $\mathbf{A}^T\beta \geq b_0$ to be well-defined, the dimension of $\mathbf{A}^T$ is $N \times (p+1)$, which is $80 \times 3$ in this case.

Writing the *i-th* row of $\mathbf{A}^T$ to be $(\mathbf{A}^T)_i = (\alpha_{1,i} \quad \alpha_{2,i} \quad \alpha_{3,i})$. Hence, $(\mathbf{A}^T)_i\beta = w_1\alpha_{1,i} + w_2\alpha_{2,i} + b\alpha_{3,i}$, and $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) = w_1X_{1,i}y_i + w_2X_{2,i}y_i + by_i$, we get:

$$\alpha_{1,i} = X_{1,i}y_i \quad ; \quad \alpha_{2,i} = X_{2,i}y_i \quad ; \quad \alpha_{3,i} = y_i$$

The ingredients for the `solve.QP` are ready. The following `R` code demonstrates how to solve the linear separable SVM optimization problem.

```
p2 <- p + 1; N <- 2 * n
Dmat <- matrix(rep(0, p2, p2), nrow = p2, ncol = p2)     ### Matrix D
diag(Dmat) <- 1; Dmat[p2,p2] <- 0
dvec <- rep(0,p2)                                        ### Vector d
Amat <- data.frame(X1=x[,1], X2=x[,2], Y=y)
Amat[, c("X1", "X2")] <- Amat[, c("X1", "X2")] * Amat$Y  ### Matrix A^T
bvec <- rep(1, N)                                        ### Vector b_0
```

We then plug in the matrices and vectors to `solve.QP`. When the $D$ matrix is used, the code complains that matrix D in quadratic function is not positive definite. To solve this problem, we add a small perturbation, i.e. $10^{-5}\mathbf{I}$ to $\mathbf{D}$, so the matrix is positive definite.

```
QP_fit <- solve.QP(Dmat + 1e-5 * diag(p2),dvec,t(Amat),bvec=bvec)
```

Now we have obtained the fitted $w_1$, $w_2$ and $b$ by `solve.QP`, and their values are:

```
w1_QP <- QP_fit$solution[1]; w2_QP <- QP_fit$solution[2]; b_QP <- QP_fit$solution[3]
w1_QP; w2_QP; b_QP
```

```
## [1] -0.9334294
```

```
## [1] -0.3849838
```

```
## [1] 2.782384
```

We obtain the indices of support vectors from the `solve.QP` module, which the Lagrangians are non-zero.

```
QP_SV = QP_fit$iact; QP_SV
```

```
## [1] 63 21 44
```

We would compare the results produced by `solve.QP` with that by `e1071` packages. Here, we conduct SVM classification using the svm module in `e1071`. We use a linear kernel, and we set the cost $C$ to be extremely large for the separable case. The $\mathbf{w}$

```
svm.fit <- svm(y ~ x, data = data.frame(x, y), type='C-classification',
               kernel='linear',scale=FALSE, cost = 1e4)
w_e1071 <- t(svm.fit$coefs) %*% svm.fit$SV; b_e1071 <- -svm.fit$rho
w_e1071[1]; w_e1071[2]; b_e1071
```

```
## [1] -0.933874
```

```
## [1] -0.3844957
```

```
## [1] 2.781953
```

The results produced by both packages are almost identical, with minuscule difference due to round-off numerical errors.

We check the support vectors computed by the `e1071` package:

```
svm.fit$index
```

```
## [1] 21 44 63
```

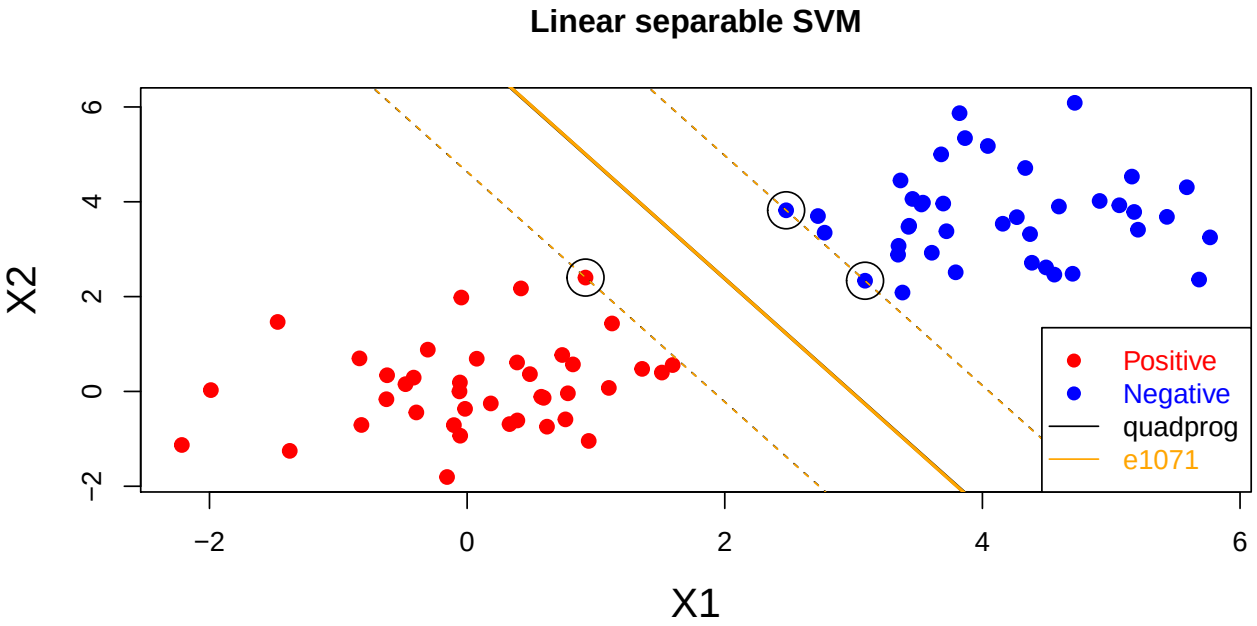
which is the same as the one computed by `solve.QP`.

The equation of the SVM hyperplanes is:

$$w_1 X_1 + w_2 X_2 + b = k \quad \Rightarrow X_2 = -\frac{w_1}{w_2} X_1 - \frac{b+k}{w_2}$$

where $k = 0$ for the separating hyperplane, and $k = \pm 1$ for the planes containing margins.

```
plot(x,col=ifelse(y>0,"red", "blue"), pch = 19, cex = 1, lwd = 2,
     xlab = "X1", ylab = "X2", cex.lab = 1.5)
abline(a= -b_QP/w2_QP, b=-w1_QP/w2_QP, col="black", lty=1, lwd = 2)
abline(a= -(b_QP+1)/w2_QP, b=-w1_QP/w2_QP, col="black", lty=2, lwd = 1)
abline(a= -(b_QP-1)/w2_QP, b=-w1_QP/w2_QP, col="black", lty=2, lwd = 1)
abline(a= -b_e1071/w_e1071[2], b=-w_e1071[1]/w_e1071[2], col="orange", lty=1, lwd = 2)
abline(a= -(b_e1071+1)/w_e1071[2], b=-w_e1071[1]/w_e1071[2], col="orange", lty=2, lwd = 1)
abline(a= -(b_e1071-1)/w_e1071[2], b=-w_e1071[1]/w_e1071[2], col="orange", lty=2, lwd = 1)
points(x[QP_SV, ], col="black", cex=3)
title("Linear separable SVM")
legend("bottomright", c("Positive","Negative", "quadprog", "e1071"), bg = "white",
       col=c("red", "blue", "black", "orange"), pch=c(19, 19, NA, NA),
       lty = c(NA, NA, 1, 1), text.col=c("red", "blue", "black", "orange"), cex = 1)
```




The support vectors are the data points circled by the black circles (3 of them in total, 2 are blue and 1 is red in color.). The black and orange solid lines are the separating hyperplanes computed by the `quadprog` and `e1071` packages respectively, and the dashed lines represent the margins. As the two packages resulted in the same result, the black and orange lines overlap with each other. The equation of the separating line is:

$$w_1X_1 + w_2X_2 + b = 0 \quad \Rightarrow \boxed{-0.93X_1 - 0.38X_2 + 2.78 = 0}$$

And the equations of the dashed lines are:

$$w_1X_1 + w_2X_2 + b = \pm 1 \quad \Rightarrow \boxed{-0.93X_1 - 0.38X_2 + 2.78 = \pm 1}$$

## Part b.

In the dual form of linear separable SVM, we maximize $\mathcal{L}_\mathcal{P}$ w.r.t. $\mathbf{a}$, with the constraints $\sum_{i=1}^N a_i y_i = 0$ and $a_i \geq 0$, for $i = 1, 2, ..., N$. Alternatively, we are minimizing the negation of $\mathcal{L}_\mathcal{P}$, i.e. $-\sum_{i=1}^N a_i + \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N a_i a_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$. This is the optimization form that `solve.QP` takes.

We formulate the problem as follows:

$$\beta^T = \begin{bmatrix} a_1, a_2, \cdots, a_N \end{bmatrix} \quad , \quad d^T = \begin{bmatrix} 1, 1, \cdots, 1 \end{bmatrix}_{1\times N} \quad \text{s.t. } -d^T\beta = -\sum_{i=1}^N a_i$$

We further define the matrices as follows: We define a $p \times N$ (*i.e.* $2 \times 80$) matrix $\mathbf{Q}$, such that:

$$\mathbf{Q}_{ik} = y_k \cdot x_{ik}, \text{ for } i = 1, 2 \text{ and } k = 1, 2, ..., 80$$

So we can define the matrix $\mathbf{D}$ of dimension $N \times N$ (*i.e.* $80 \times 80$), such that:

$$\mathbf{D} = \mathbf{Q}^T\mathbf{Q} \quad ; \quad \mathbf{D}_{mn} = (y_m \cdot x_{im})^T(y_n \cdot x_{in}) \quad ; \quad \text{s.t. } \frac{1}{2}\beta^T\mathbf{D}\beta = \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N a_i a_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}$$

For the constraints, we have $N + 1$ constraints, they are $\sum_{i=1}^N a_i y_i = 0$, and $a_i \geq 0$, for $i = 1, 2, ..., N$.

Therefore, we formulate the constraining matrices and vectors as follows:

$$b_0^T = \left[0, 0, \cdots, 0\right]_{1\times(N+1)} \quad , \quad \mathbf{A} = \begin{bmatrix} y_1 & 1 & 0 & \cdots & 0 \\ y_2 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ y_N & 0 & 0 & \cdots & 1 \end{bmatrix}_{N\times(N+1)} \quad \text{s.t. } \mathbf{A}^T\beta = \begin{bmatrix} \sum_{i=1}^N a_i y_i \\ a1 \\ a2 \\ \vdots \\ a_N \end{bmatrix}_{(N+1)\times 1} \geq b_0$$

In other words, $\mathbf{A}$ is constructed as follows: the 1st column being the responses of all data points, $y_1, y_2, ..., y_N$, and the 2nd to $N+1$ columns are the identity matrix of $N \times N$. Here, we build the system of the equations and solve them by `solve.QP`. Again, as the matrix $\mathbf{D}$ is not positive definite, we introduce a tiny perturbation to it. The parameter `meq` is set to be 1 since the first constraining equation is an equality constraint.

```
Qmat_dual <- sapply(1:N, function(i) y[i]*t(x)[,i])    # Matrix Q
Dmat_dual <- t(Qmat_dual) %*% Qmat_dual                # Matrix D
dvec_dual <- matrix(1, nrow = N)                       # Vector d
bvec_dual <- rbind(matrix(0, nrow = 1, ncol = 1), matrix(0, nrow = N, ncol = 1)) # Vector b0
Amat_dual <- t(rbind(matrix(y, nrow=1, ncol=N), diag(nrow=N)))   # Matrix A
dual_sol <- solve.QP(Dmat_dual + 1e-5*diag(N), dvec_dual, Amat_dual, bvec_dual, meq=1,
                     factorized=FALSE)
```

This solves the $a_i$'s. Many of them are close to zero, but not quite due to round-off errors, as they should as they are not support vectors, which theoretically should have zero $a_i$'s. Three of them have significantly non-zero $a_i$'s, which are the support vectors.

```
a_dual <- dual_sol$solution; dual_SV <- which(abs(a_dual) > 1e-4); dual_SV
```

```
## [1] 21 44 63
```

The indices coincide with those found in part a, whose $a_i$'s are as follows:

```
a_dual[dual_SV]
```

```
## [1] 0.5097718 0.2820910 0.2276800
```

As a result, $\mathbf{w}$ is found by:

```
dual_w <- t(a_dual * y) %*% x
```

$b$ is estimated by $\hat{b} = -\frac{\max_{i,y_i=-1} \mathbf{x_i}^T\mathbf{w} + \min_{i,y_i=1} \mathbf{x_i}^T\mathbf{w}}{2}$. To select the appropriate support vectors, we split the support vectors with classified groups, and select the maximum when $y_i = -1$, and minimum when $y_i = 1$. We write a function to perform this task.

```
est_intercept <- function(SV, x, w, y) {
  y_SV <- y[SV]; x_SV <- x[SV,]
  xw_product <- x_SV %*% t(w)
  xw_posarr <- xw_product[which(y_SV > 0)]
  xw_negarr <- xw_product[which(y_SV < 0)]
  xw_pos <- xw_posarr[which.max(xw_posarr)]
  xw_neg <- xw_negarr[which.min(xw_negarr)]
  return(-0.5 * (xw_pos + xw_neg))
}
```

The results computed from the dual formulation are:

```
dual_b <- est_intercept(dual_SV, x, dual_w, y)
dual_w; dual_b
```

```
##           [,1]      [,2]
## [1,] -0.933426 -0.384982
```

```
## [1] 2.782373
```

which are consistent to those computed in part a, and the result of `e1071`. The plot and the equations are identical to those in part a.

## Part c.

For the non separable SVM, we have the same Lagrangian to optimize as the separable dual case (part b). Hence, we have identical constructions of the objective function vectors and matrices, i.e. $\beta$, $d$, $\mathbf{Q}$ and $\mathbf{D}$. We have constraint equations $\sum_{i=1}^{N} a_i y_i = 0; \quad 0 \leq a_i \leq C$ , for all $i$, hence we construct as follows,

$$b_0^T = \left[\overbrace{0, 0, \cdots, 0}^{N+1}, \overbrace{-C, -C, \cdots, -C}^{N}\right]_{1\times(2N+1)} \quad , \quad \mathbf{A} = \left[\mathbf{A_{11}}_{N\times(N+1)} \quad \mathbf{A_{12}}_{N\times N}\right]_{N\times(2N+1)}$$

$$\text{where } \mathbf{A_{11}} = \begin{bmatrix} y_1 & 1 & 0 & \cdots & 0 \\ y_2 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ y_N & 0 & 0 & \cdots & 1 \end{bmatrix}_{N\times(N+1)} \quad , \quad \mathbf{A_{12}} = -\mathbf{I}_{N\times N}$$

$$\text{s.t. } (\mathbf{A}^T\beta)^T = \left[\left(\sum_{i=1}^{N} a_i y_i\right), a_1, a_2, \cdots, a_N, -a_1, -a_2, \cdots, -a_N\right]_{1\times(2N+1)} \geq b_0^T$$

As we constrain $a_i \leq C$, this is equivalent to $-a_i \geq -C$. We generate a set of nonseparable data.

```
set.seed(40); n <- 10; p <- 2; N <- 2 * n
# 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 <- matrix(c(rep(1,n),rep(-1,n)))
```

We write a function for solving the non-separable SVM for `solve.QP`, we reuse the code in part d.

```
nonsep_SVM <- function(N, C, x, y) {
  Qmat_dual <- sapply(1:N, function(i) y[i]*t(x)[,i])   # Matrix Q
  Dmat_dual <- t(Qmat_dual) %*% Qmat_dual           # Sub Matrix D11
  dvec_dual <- matrix(1, nrow = N)                    # Vector d
  bvec_dual <- matrix(0, nrow = (N + 1), ncol = 1) # Vector b0
  bvec_dual <- rbind(bvec_dual, matrix(-C, nrow = N, ncol = 1))
  Amat_dual <- t(rbind(matrix(y, nrow=1, ncol=N), diag(nrow=N), -diag(nrow=N)))   # Matrix A
  dual_sol <- solve.QP(Dmat_dual + 1e-3*diag(N), dvec_dual, Amat_dual, bvec_dual, meq=1,
                       factorized=FALSE)                          # Vector b_0
  return(dual_sol)
}
```

We set a value of $C = 1$.

```
dual_sol <- nonsep_SVM(N, 1, x, y)
```

We find the support vectors, $\mathbf{w}$ and $b$. We have to find $b$ by identifying the support vector with the maximum weight of the two classes, i.e. the support vectors that lie on the margin on the correct side.

```
a_dual <- dual_sol$solution; dual_SV <- which(abs(a_dual) > 1e-4)
dual_w <- t(a_dual * y) %*% x; dual_b <- est_intercept(dual_SV, x, dual_w, y)
dual_SV; dual_w; dual_b
```

```
##  [1]  1  3  5  8  9 12 15 17 19 20
```

```
##             [,1]       [,2]
## [1,] -0.5399922 -0.8279698
```

```
## [1] 1.20024
```

We confirm this with the one fitted by the e1071 package.

```
svm.fit <- svm(y ~ x, data = data.frame(x, y), type='C-classification', kernel='linear',
               scale=FALSE, cost = 1)
w <- t(svm.fit$coefs) %*% svm.fit$SV; b <- -svm.fit$rho;
svm.fit$index; w; b
```

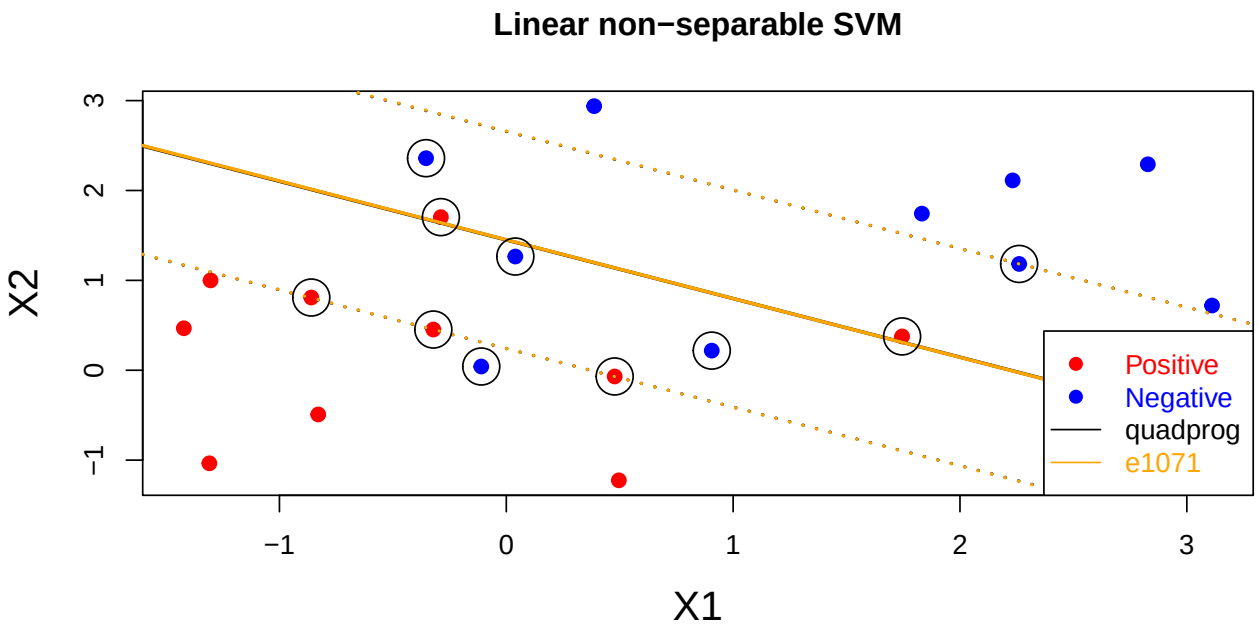
```
##  [1]  1  3  5  8  9 12 15 17 19 20
```

```
##              x1         x2
## [1,] -0.5407657 -0.8268247
```

```
## [1] 1.200575
```

The results of the two packages again agree up to round-off errors. Hence, the equation of the separating line is: $\boxed{-0.54X_1 - 0.83X_2 + 1.20 = 0}$.

```
plot(x,col=ifelse(y>0,"red", "blue"), pch = 19, cex = 1, lwd = 2,
     xlab = "X1", ylab = "X2", cex.lab = 1.5)
abline(a= -dual_b/dual_w[1,2], b=-dual_w[1,1]/dual_w[1,2], col="black", lty=1, lwd = 2)
abline(a= (-dual_b-1)/dual_w[1,2], b=-dual_w[1,1]/dual_w[1,2], col="black", lty=3, lwd = 2)
abline(a= (-dual_b+1)/dual_w[1,2], b=-dual_w[1,1]/dual_w[1,2], col="black", lty=3, lwd = 2)
abline(a= -b/w[1,2], b=-w[1,1]/w[1,2], col="orange", lty=1, lwd = 2)
abline(a= (-b-1)/w[1,2], b=-w[1,1]/w[1,2], col="orange", lty=3, lwd = 2)
abline(a= (-b+1)/w[1,2], b=-w[1,1]/w[1,2], col="orange", lty=3, lwd = 2)
points(x[dual_SV,], col="black", cex=3)
title("Linear non-separable SVM")
legend("bottomright", c("Positive","Negative", "quadprog", "e1071"), bg = "white",
       col=c("red", "blue", "black", "orange"), pch=c(19, 19, NA, NA),
       lty = c(NA, NA, 1, 1), text.col=c("red", "blue", "black", "orange"), cex = 1)
```




Again, the separating lines computed from the two packages overlap with each other. The support vectors are circled by the black circles.

## Part d.

We load in the South Africa Heart Disease data. Data preprocessing: `famhist` is categorical, we have to convert it to 0 or 1 so the code works. Also, we convert the 0's in `chd` to -1 so SVM would work.

```
library(ElemStatLearn)
data(SAheart)
SAheart[,5] <- sapply(SAheart[,5],switch,"Absent"=0,"Present"=1)
SAheart["chd"][SAheart["chd"] == 0] <- -1
```

We have 9 covariates in this dataset, we use all of them to predict `chd`. The following code visualizes the pair-wise scatter plots for all combinations of variables and responses. There are strong correlations among some pairs of the variables, but this is not the main concern of the problem. Interested reader can uncomment this code to visualize the plot. The response `chd` is non-separable with each variable.

```
#pairs(SAheart)
```

We divide the dataset into 10-folds for cross validation (CV), this is to select the best tuning parameter $C$. We calculate the test error by the fraction of misclassified data to the number of test data, and taking average over all $k$-folds. A sequence of $C$ values are used for fitting, ranging from 0.001 to 10. The best model is found by searching for the model that gives the minimum test error. We compare our code with the `e1071` package.

```
set.seed(30); n <- dim(SAheart)[1]; p <- 9; K <- 10
SAheart <- SAheart[sample(n),]   # Shuffle the data
SAheart["CVfold"] <- cut(seq(1,n),breaks=K,labels=FALSE)   # Create 10 equally size folds
### Perform a 10-fold CV.
C_par <- c(0.01,0.03,0.1,0.3,1)
C_len <- length(C_par)
test_err <- matrix(0., nrow = C_len, ncol = K)   # A matrix to store the test errors
testsvm_err <- matrix(0., nrow = C_len, ncol = K)   # A matrix to store the test errors
for (j in 1:C_len) {
  print(j)
  for (i in 1:K) {
    train_ind <- which(SAheart["CVfold"] != i); test_ind <- which(SAheart["CVfold"] == i)
    x_train <- data.matrix(SAheart[,1:p])[train_ind,]
    y_train <- as.matrix(SAheart["chd"])[train_ind,]
    x_test <- data.matrix(SAheart[,1:p])[test_ind,]
    y_test <- as.matrix(SAheart["chd"])[test_ind,]
    # Train the SVM model.
    n_train <- dim(x_train)[1]
    SA_sol <- nonsep_SVM(n_train, C_par[j], x_train, y_train)
    a_SA <- SA_sol$solution; SA_SV <- which((abs(a_SA) > 1e-4) & (abs(a_SA) < 0.999999))
    SA_w <- t(a_SA * y_train) %*% x_train
    SA_b <- est_intercept(SA_SV, x_train, SA_w, y_train)
    # Test the trained model.
    n_test <- dim(x_test)[1]
    y_pred <- x_test %*% t(SA_w) + SA_b
    y_pred[which(y_pred < 0)] <- -1
    y_pred[which(y_pred >= 0)] <- 1
    test_err[j,i] <- length(which((y_pred-y_test) != 0)) / n_test # Test error percentage.
    # e1071 package
    svm.fit <- svm(y_train ~ ., data = data.frame(x_train, y_train), type='C-classification',
                   kernel='linear', scale=FALSE, cost = C_par[j])
    w <- -t(svm.fit$coefs) %*% svm.fit$SV;
    b <- svm.fit$rho;
```

```
    print(w); print(b)
    print(SA_w); print(SA_b)
    y_svmpred <- x_test %*% t(w) + b
    y_svmpred[which(y_svmpred < 0)] <- -1
    y_svmpred[which(y_svmpred >= 0)] <- 1
    testsvm_err[j,i] <- length(which((y_svmpred-y_test) != 0)) / n_test # Test error percentage.
  }
}
```

The $\mathbf{w}$ and $\mathbf{b}$ vectors calculated by the two approaches are slightly off, but that does not alter the conclusion from the SVM. Having computed the matrix of test errors, we average across the columns to obtain the averaged test error for each $C$. The best $C$ and the corresponding test error are shown as follows. We compare the results given by our code and the `e1071` package.

```
QP_mean <- rowMeans(test_err); SVM_mean <- rowMeans(testsvm_err)
QP_min_err_k <- which.min(QP_mean); SVM_min_err_k <- which.min(SVM_mean)
C_par[QP_min_err_k]; QP_mean[QP_min_err_k]
```

```
## [1] 0.03
```

```
## [1] 0.2834875
```

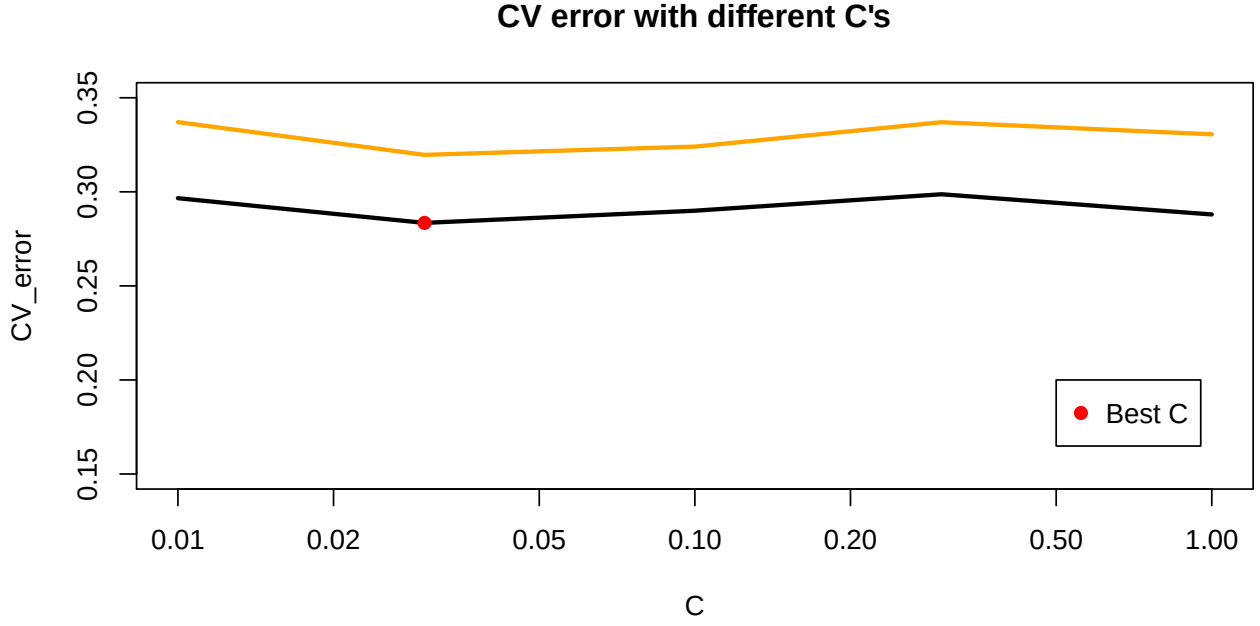
```
C_par[SVM_min_err_k]; SVM_mean[SVM_min_err_k]
```

```
## [1] 0.03
```

```
## [1] 0.3196577
```

The graph showing the CV error from our code is shown as follows. The orange and black solid lines represent the result from `e1071` and `quadprog` packages respectively.

```
plot(C_par, QP_mean, log="x", xlab = "C", ylab = "CV_error", col = "black", type = "l",
     lwd = 2.5, main = "CV error with different C's", ylim = c(0.15,0.35))
lines(C_par, SVM_mean, col = "orange", type = "l", lwd = 2.5)
points(x = C_par[QP_min_err_k], y = QP_mean[QP_min_err_k], col = "red", pch = 19)
legend(x = 0.5, y = 0.2, legend = "Best C", col = "red", pch = 19)
```



## Question 2.

Without loss of generality, assume $\xi_1 < \xi_2 < \cdots < \xi_K$. Taking the first, second and third derivatives of $f(X)$:

$$f(X) = \sum_{j=0}^{3} \beta_j X^j + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3 = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3$$

$$f'(X) = \sum_{j=1}^{3} j\beta_j X^{j-1} + 3\sum_{k=1}^{K} \theta_k (X - \xi_k)_+^2 = \beta_1 + 2\beta_2 X + 3\beta_3 X^2 + 3\sum_{k=1}^{K} \theta_k (X - \xi_k)_+^2$$

$$f''(X) = \sum_{j=2}^{3} j(j-1)\beta_j X^{j-2} + 6\sum_{k=1}^{K} \theta_k (X - \xi_k)_+ = 2\beta_2 + 6\beta_3 X + 6\sum_{k=1}^{K} \theta_k (X - \xi_k)_+$$

$$f'''(X) = \sum_{j=3}^{3} j(j-1)(j-2)\beta_j X^{j-3} + 6\sum_{k=1}^{K} \theta_k = 6\beta_3 + 6\sum_{k=1}^{K} \theta_k$$

For $X < \xi_1$, $f'(X) = \beta_1 + 2\beta_2 X + 3\beta_3 X^2$ (Since $(X - \xi_k)_+ = 0$, for $k = 1, 2, ..., K$). Since $f(X)$ is linear for $X \leq \xi_1$, $f'(X)$ is a constant, which does not depend on $X$ as long as $X \leq \xi_1$. So the coefficients for power of $X$'s in $f'(X)$ would be zero. $\boxed{\textit{i.e. } \beta_2 \text{ and } \beta_3 \text{ are both 0.}}$

At $X = \xi_1$, since $f''(\xi_1)_- = f''(\xi_1)_+ = 0$, hence $f'''(\xi_1)_- = f'''(\xi_1)_+ = 0$. Since $\beta_3 = 0$, from the above expressions, $\boxed{\sum_{k=1}^{K} \theta_k = 0}$.

At $X = \xi_K$, since $f''(\xi_K)_- = f''(\xi_K)_+ = 0$, we have

$$f''(\xi_K) = 6\sum_{k=1}^{K} \theta_K (\xi_K - \xi_k) = 0 \quad \Rightarrow \quad \boxed{\sum_{k=1}^{K} \theta_k \xi_k = \xi_K \sum_{k=1}^{K} \theta_k = \xi_K(0) = 0}$$

For general k,

$$\begin{aligned}
\alpha_k(d_k(X) - d_{K-1}(X)) &= \theta_k(\xi_K - \xi_k)\left[\frac{(X - \xi_k)_+^3 - (X - \xi_K)_+^3}{\xi_K - \xi_k} - \frac{(X - \xi_{K-1})_+^3 - (X - \xi_K)_+^3}{\xi_K - \xi_{K-1}}\right] \\
&= \theta_k \frac{(\xi_K - \xi_{K-1})\left[(X - \xi_k)_+^3 - (X - \xi_K)_+^3\right] - (\xi_K - \xi_k)\left[(X - \xi_{K-1})_+^3 - (X - \xi_K)_+^3\right]}{\xi_K - \xi_{K-1}} \\
&= \theta_k \frac{(\xi_K - \xi_{K-1})(X - \xi_k)_+^3 + \xi_{K-1}(X - \xi_K)_+^3 - (\xi_K - \xi_k)(X - \xi_{K-1})_+^3 - \xi_k(X - \xi_K)_+^3}{\xi_K - \xi_{K-1}}
\end{aligned}$$

Consider the regions $\xi_1 \leq X < \xi_{K-1}$, i.e. $1 \leq k \leq K-2$, only the first term in the numerator ($(\xi_K - \xi_{K-1})(X - \xi_k)_+^3$) is non-zero. Hence, $\alpha_k(d_k(X) - d_{K-1}(X)) = \theta_k(X - \xi_k)_+^3$.

For $k = K-1$, $\alpha_{K-1}(d_{K-1}(X) - d_{K-1}(X)) = 0 = \theta_{K-1}(X - \xi_{K-1})_+^3$.

For $k = K$,

$$\begin{aligned}
&\alpha_K(d_K(X) - d_{K-1}(X)) \\
&= \theta_K \frac{(\xi_K - \xi_{K-1})(X - \xi_K)_+^3 + \xi_{K-1}(X - \xi_K)_+^3 - (\xi_K - \xi_K)(X - \xi_{K-1})_+^3 - \xi_K(X - \xi_K)_+^3}{\xi_K - \xi_{K-1}} \\
&= \theta_K \frac{(\xi_K - \xi_{K-1})(X - \xi_K)_+^3 - (\xi_K - \xi_{K-1})(X - \xi_K)_+^3}{\xi_K - \xi_{K-1}} = 0 = \theta_K(X - \xi_K)_+^3
\end{aligned}$$

Therefore,

$$\sum_{k=1}^{K} \theta_k (X-\xi_k)_+^3 = \sum_{k=1}^{K-2} \theta_k (X-\xi_k)_+^3 + \theta_{K-1}(X-\xi_{K-1})_+^3 + \theta_K (X-\xi_K)_+^3 = \sum_{k=1}^{K-2} \theta_k (X-\xi_k)_+^3$$
$$= \sum_{k=1}^{K-2} \alpha_k (d_k(X) - d_{K-1}(X))$$

Together with the fact that $\beta_2 = \beta_3 = 0$, we write $f(X)$ as:

$$\boxed{f(X) = \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \alpha_k (d_k(X) - d_{K-1}(X))}$$

where

$$d_k(X) = \frac{(X-\xi_k)_+^3 - (X-\xi_K)_+^3}{\xi_K - \xi_k} \quad \text{and} \quad \alpha_k = \theta_k(\xi_K - \xi_k).$$

# Question 3.

We analyze the `Mass Shootings Dataset Ver 2.csv` file from Kaggle. We state the research problem of this dataset later. We firstly perform data preprocessing. We load in the four columns, namely `Fatalities`, `Total.victims`, `Latitude` and `Longitude`. We check is there is any missing values (`NA`) in the columns.

```
MS_df <- read.csv("Mass Shootings Dataset Ver 2.csv", header = TRUE, sep = ",")
MS_df <- MS_df[c("Fatalities","Total.victims","Latitude","Longitude")]
n <- dim(MS_df)[1]          # Number of rows.
which(is.na(MS_df["Fatalities"]))
```

```
## integer(0)
```

```
which(is.na(MS_df["Total.victims"]))
```

```
## integer(0)
```

```
which(is.na(MS_df["Longitude"]))
```

```
##  [1]   2   3   4   5   6   7   8   9  10  11  49  55  78  80 106 152 154
```

```
which(is.na(MS_df["Latitude"]))
```

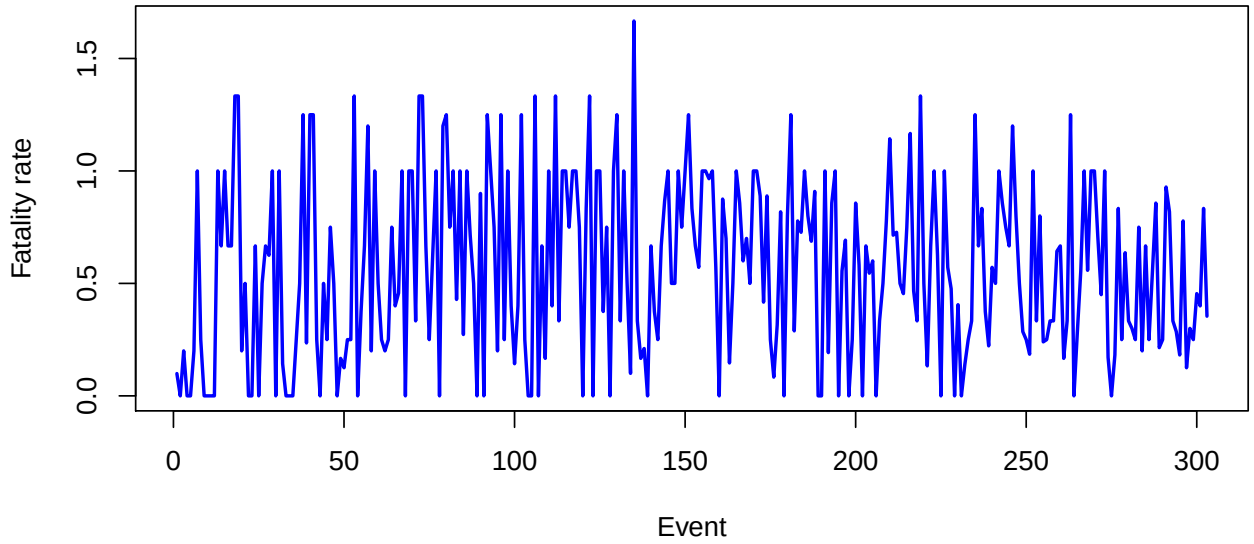
```
##  [1]   2   3   4   5   6   7   8   9  10  11  49  55  78  80 106 152 154
```

Hence, there is no missing values for `Fatalities` and `Total.victims`, but some for the coordinates. However, there are only 17 missing values out of 320 data points, so we drop those 17 entries and have 303 data points left.

```
rowkeep <- which(is.na(MS_df$Longitude) == FALSE)
MS_df <- as.matrix(MS_df)
MS_df <- MS_df[rowkeep,]
n <- length(MS_df[,1])
```

We calculate the fatality rate, i.e. `Fatalities` / `Total.victims`. And we visualize the fatality rate to spot if there is any strange behavior.

```
MS_y <- as.matrix(MS_df[,1] / MS_df[,2])
plot(seq(1,n), MS_y, xlab = "Event", ylab = "Fatality rate", col = "blue", type = "l", lwd = 2)
```

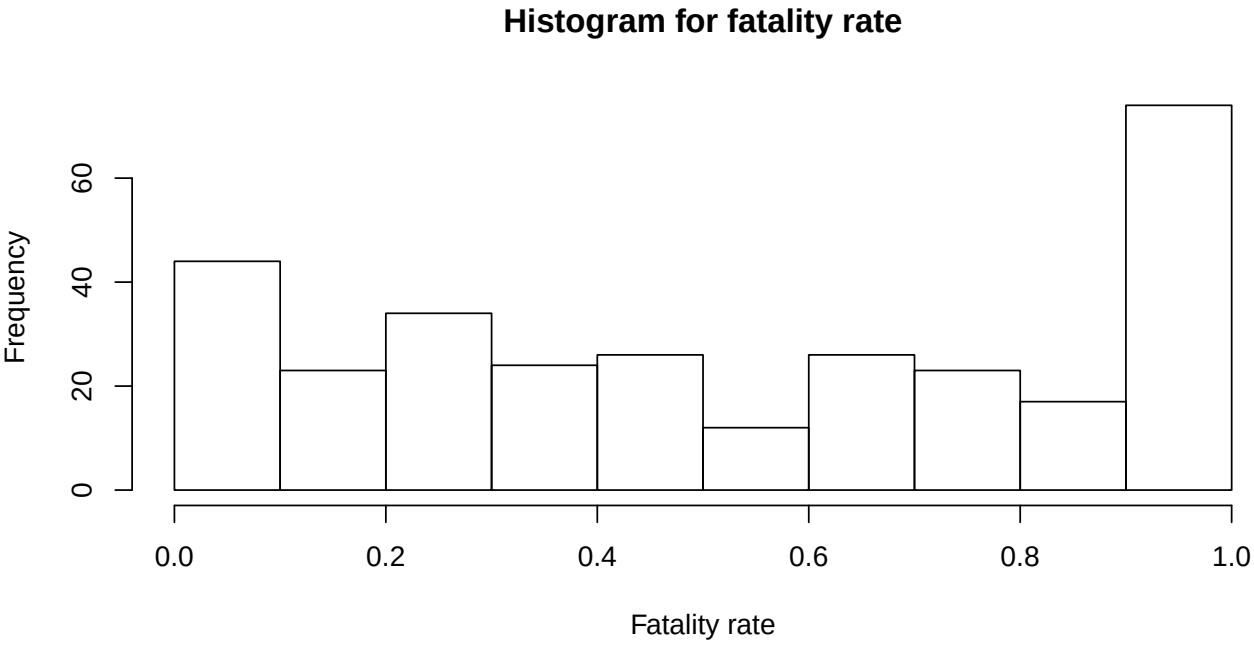



Most of the data is within 0.0 and 1.0. There are some of them which exceed 1.0. We check the difference between `Fatalities` and `Total.victims`, i.e. `Fatalities - Total.victims`

```
MS_check <- MS_df[which(MS_y > 1),]
MS_diff <- MS_check[,1] - MS_check[,2]
MS_y[which(MS_y > 1),] <- 1          # Cap the fatality rate to 1.
MS_x <- cbind(MS_df[,3], MS_df[,4])
```

Most of the differences are 1, except for one case where the number is 2. We assume in these cases that the murderers committed suicide. In the case where difference is 2, there were two murderers and both of them committed suicide. However, even in the data points where fatality rate is lower than 1, the murderers could have committed suicide as well and there is not enough information in the dataset to differentiate those cases. Hence, we keep the cases where fatality rate is higher than 1, and cap those rates to 1. The reader should keep that in mind. We then visualize the distribution of fatality rate.

```
hist(MS_y, main="Histogram for fatality rate", xlab = "Fatality rate")
```



```r
length(which(MS_y <= 0.5)); length(which(MS_y > 0.5))
```

```
## [1] 151
```

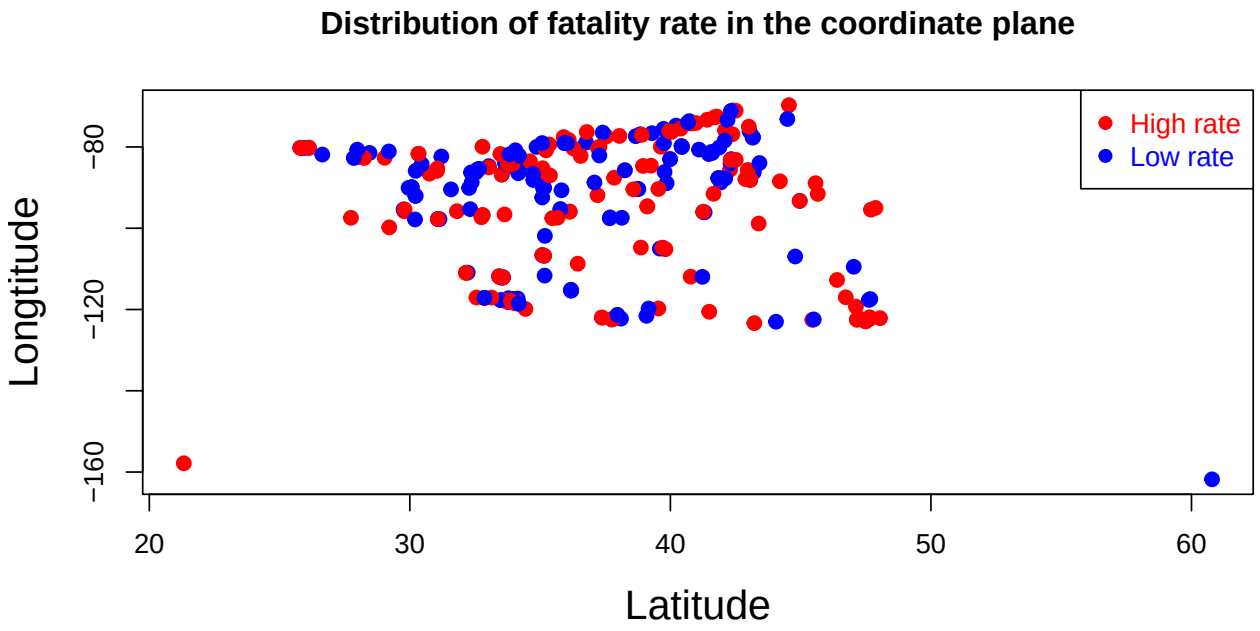
```
## [1] 152
```

We see if we divide the fatality rate into two groups at 0.5, the high and low fatality rates have comparable number of data points. **Here comes our research problem: Is location an indicative predictor for the fatality rate of the mass shooting**. We define high fatality rate as those rates $\geq 0.5$, and we label it as $+1$; whereas the labels of those low fatality rate as $-1$. We approach this problem with SVM.

```r
MS_y[which(MS_y <= 0.5)] <- -1
MS_y[which(MS_y > 0.5)] <- 1
```

Hence, we visualize the distribution of fatality rate in the coordinate plane.

```r
plot(MS_x,col=ifelse(MS_y>0,"red", "blue"), pch = 19, cex = 1, lwd = 2,
     xlab = "Latitude", ylab = "Longtitude", cex.lab = 1.5)
title("Distribution of fatality rate in the coordinate plane")
legend("topright", c("High rate","Low rate"), bg = "white", col=c("red", "blue"),
       pch=c(19, 19), text.col=c("red", "blue"), cex = 1)
```




Clearly this dataset is not linearly separable. We try using the non separable SVM with radial kernel. As a remark, we do not wish to witness another mass shooting. The prediction accuracy of this model is just for illustrative purpose on how SVM performs on this dataset. Following Q.1., we perform a 10-fold CV to tune parameter $C$ to search for the best model.

```r
set.seed(34)
n <- dim(MS_x)[1]; p <- 2; K <- 10
MS_dfcor <- data.frame(MS_x, MS_y)
MS_dfcor <- MS_dfcor[sample(n),]   # Shuffle the data
MS_dfcor["CVfold"] <- cut(seq(1,n),breaks=K,labels=FALSE)   # Create 10 equally size folds
C_par <- c(0.01,0.03,0.1,0.3,1,3,10,30,100,300,1000,3000)
C_len <- length(C_par)
MSpred_err <- matrix(0., nrow = C_len, ncol = K)   # A vector to store the prediction errors
for (j in 1:C_len) {
  for (i in 1:K) {
```

```
    train_ind <- which(MS_dfcor["CVfold"] != i); test_ind <- which(MS_dfcor["CVfold"] == i)
    MS_x_train <- data.matrix(MS_dfcor[,1:p])[train_ind,]
    MS_y_train <- as.matrix(MS_dfcor["MS_y"])[train_ind,]
    MS_x_test <- data.matrix(MS_dfcor[,1:p])[test_ind,]
    MS_y_test <- as.matrix(MS_dfcor["MS_y"])[test_ind,]
    svm.fit <- svm(MS_y_test ~ ., data = data.frame(MS_x_test, MS_y_test), type='C-classification',
               kernel='radial', scale=FALSE, cost = C_par[j])
    n_test <- dim(MS_x_test)[1]
    MSpred_err[j,i] <- length(which((svm.fit$fitted != MS_y_test) != 0)) / n_test
  }
}
```

We then calculate the averaged CV errors for different $C$'s. And we report the best fitted $C$'s and the corresponding test error.

```
MS_mean <- rowMeans(MSpred_err)
MS_min_err_k <- which.min(MS_mean)
C_par[MS_min_err_k]; MS_mean[MS_min_err_k]
```

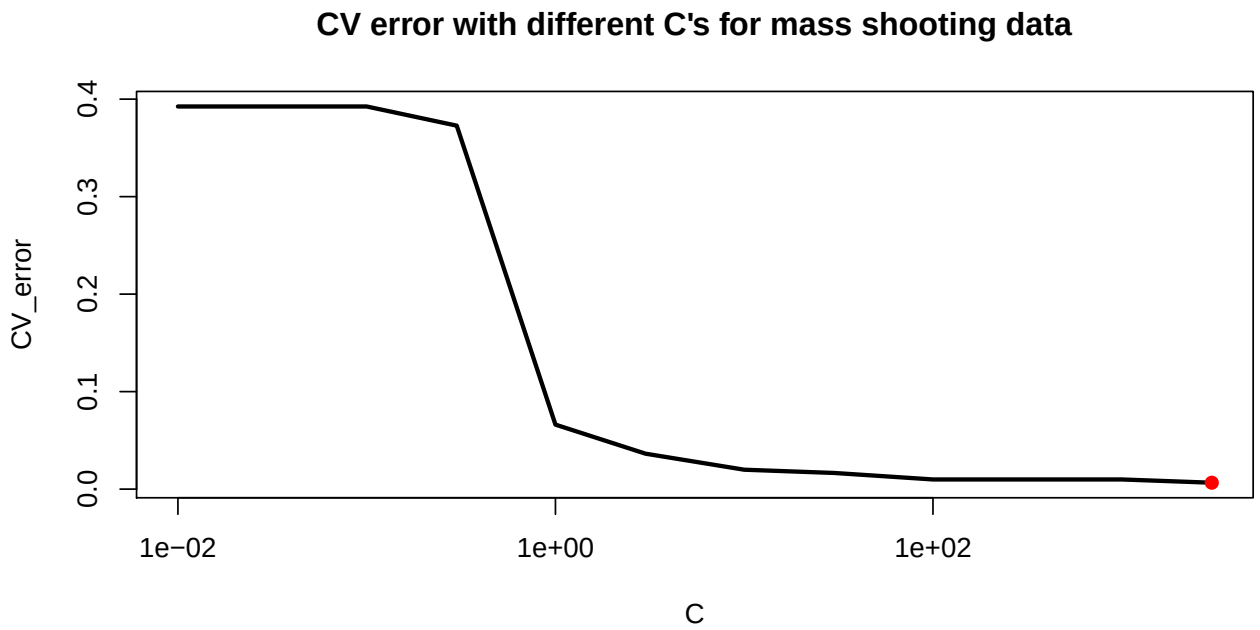
```
## [1] 3000
```

```
## [1] 0.00655914
```

The best fitted model shows test error of less than 1%! For non-linear SVM, we can penalize the high complexity model by calculating $\min_f \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + C||f||_H^2$, where $f$ belongs to the kernel Hilbert space $H$. The calculation of penalizing nonlinear SVM is beyond the scope of this report. We stick with the best fitted model as found by the SVM with radial kernel.

The graph showing the CV error from our code is shown as follows. The red point represents the best fitted $C$.

```
plot(C_par, MS_mean, log="x", xlab = "C", ylab = "CV_error", col = "black", type = "l",
     lwd = 2.5, main = "CV error with different C's for mass shooting data")
points(x = C_par[MS_min_err_k], y = MS_mean[MS_min_err_k], col = "red", pch = 19)
```




**Conclusion: This model can predict up to 1% accuracy on the fatality rate given the coordinates of the incident.**