STAT 542: Homework 8

Spring 2022, by Kamila Makhambetova (kamilam3)

Due: Thursday, March 31, 11:59 PM CT

Instruction

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to compass2g. No email or hardcopy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat542/homework.html), please refer to the course website.

- What is expected for the submission to **Gradescope**
  - You are required to submit one rendered **PDF** file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file generated by a `.Rmd` file. `.html` format cannot be accepted.
  - Please follow the instructions on Gradescope to select corresponding PDF pages for each question.
- Please note that your homework file is a **PDF** report instead of a messy collection of R codes. This report should **include**:
  - Your Name and NetID. (Replace `Ruoqing Zhu (rqzhu)` by your name and NetID if you are using this template).
  - Make all of your `R` code chunks visible for grading.
  - Relevant outputs from your `R` code chunks that support your answers.
  - Provide clear answers or conclusions for each question. For example, you could start with `Answer: I fit SVM with the following choice of tuning parameters ...`
  - Many assignments require your own implementation of algorithms. **Basic comments are strongly encouraged** to explain the logic to our graders. However, line-by-line code comments are unnecessary.
- Requirements regarding the `.Rmd` file.
  - You do **NOT** need to submit `Rmd` files. However, your PDF file should be rendered directly from it.

- Make sure that you **set random seeds** for simulation or randomized algorithms so that the results are reproducible. If a specific seed number is not provided in the homework, you can consider using your NetID.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.

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.

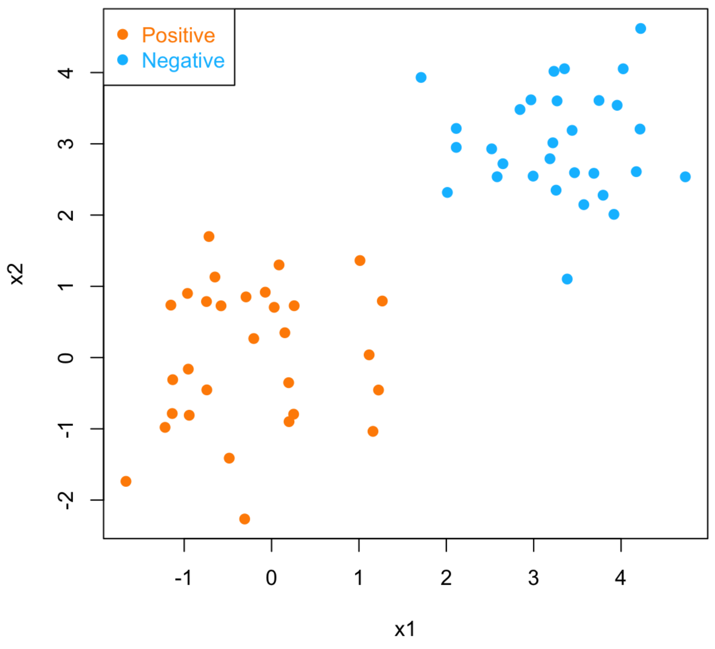
Question 1 [50 Points] Sovling SVM using Quadratic Programming

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:

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

where b 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.

```
  # this is how I generate the data
  # set.seed(3)
  # 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)))
  #
  # train = data.frame(x1 = x[, 1], x2 = x[, 2], y = y)
  # write.csv(train, "SVM-Q1.csv", row.names = FALSE)

  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", y
lab = "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

$$\begin{aligned} \underset{\beta_0, \boldsymbol{\beta}}{\text{minimize}} \quad & \frac{1}{2} \|\boldsymbol{\beta}\|^2 \\ \text{subject to} \quad & y_i \left(x_i^\top \boldsymbol{\beta} + \beta_0 \right) \geq 1, \ \text{ for } i = 1, \ldots, n \end{aligned}$$

Perform the following:

- Let $b = (\beta_0, \boldsymbol{\beta})$ in the `solve.QP()` function. Properly define $\mathbf{D}$, d, $\mathbf{A}$ and b_0 corresponding to this b for the linearly separable SVM primal problem.
- Calculate the decision function by solving this optimization problem with the `solve.QP()` function.
- Report our β_0 and $\boldsymbol{\beta}$
- 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 $\mathbf{D}$ to be positive definite, while it is not true in our case. To address this problem, **add** 10^{-10} **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.

```r
library(quadprog)

#create matrix D
Dmat=matrix(0,3,3)
Dmat[1,1]=10^{-10}
Dmat[2,2]=1
Dmat[3,3]=1

#create vector d
dvec=rep(0,3)

#create A matrix
Amat=matrix(0,nrow(x),3)
Amat[,1]=y
Amat[,2]=y*x[,1]
Amat[,3]=y*x[,2]

##create vector b
bvec=rep(1,60)

#use solve.QP to find betas
b=solve.QP(Dmat, dvec, t(Amat), bvec)$solution
b0=b[1]
beta=rep(0,2)
beta=b[2:3]

#report betas
print(paste0("b0 = ",b0))
```

```
## [1] "b0 = 3.41923843996842"
```

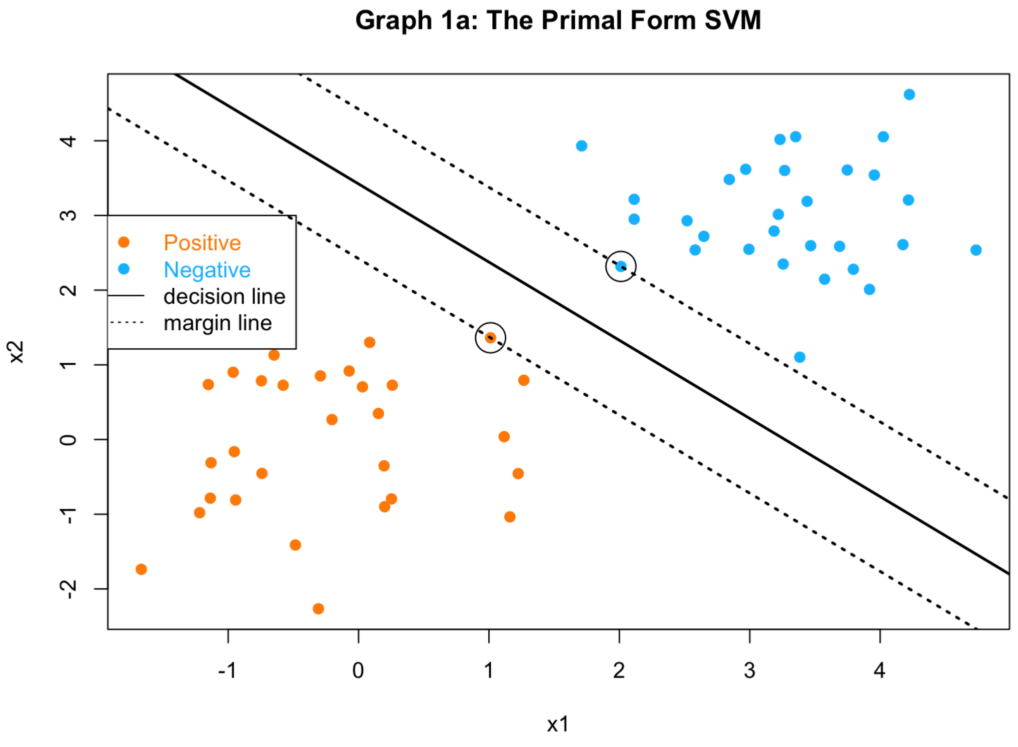
```r
print("Beta:")
```

```
## [1] "Beta:"
```

```r
print(beta)
```

```
## [1] -1.0457306 -0.9990794
```

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


 #decision line
abline(a= -b0/b[3], b=-b[2]/b[3], col="black", lty=1, lwd = 2)


# legend
op <- par(cex = 1)
legend(-2,3, c("Positive","Negative","decision line", "margin line"), col=c("da
rkorange", "deepskyblue", "black", "black"),
         pch=c(19, 19, NA, NA), lty=c(NA,NA,1,3),text.col=c("darkorange", "deep
skyblue", "black", "black"))


# mark the support vectors. The points were found by looking at graph and table
of x
points(x[49, 1], x[49, 2], col="black", cex=3)
points(x[28, 1], x[28, 2], col="black", cex=3)


# the two margin lines
abline(a= (-b0-1)/b[3], b=-b[2]/b[3], col="black", lty=3, lwd = 2)
abline(a= (-b0+1)/b[3], b=-b[2]/b[3], col="black", lty=3, lwd = 2)
title("Graph 1a: The Primal Form SVM")
```



b) [25 points] The Dual Form

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

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} \quad & \alpha_i \geq 0, \ \text{for } i = 1, \ldots, n \\ \text{and} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

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.
- Note: 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.

You need to report * A table including $\beta_0, \beta_1, \beta_2)$ 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 ℓ_1 norm of $\beta_{Q1a} - \beta_{Q1b}$, where β_{Q1a} and β_{Q2b} are the 3-dimensional solution obtained in Q1a and Q1b, respectively.

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

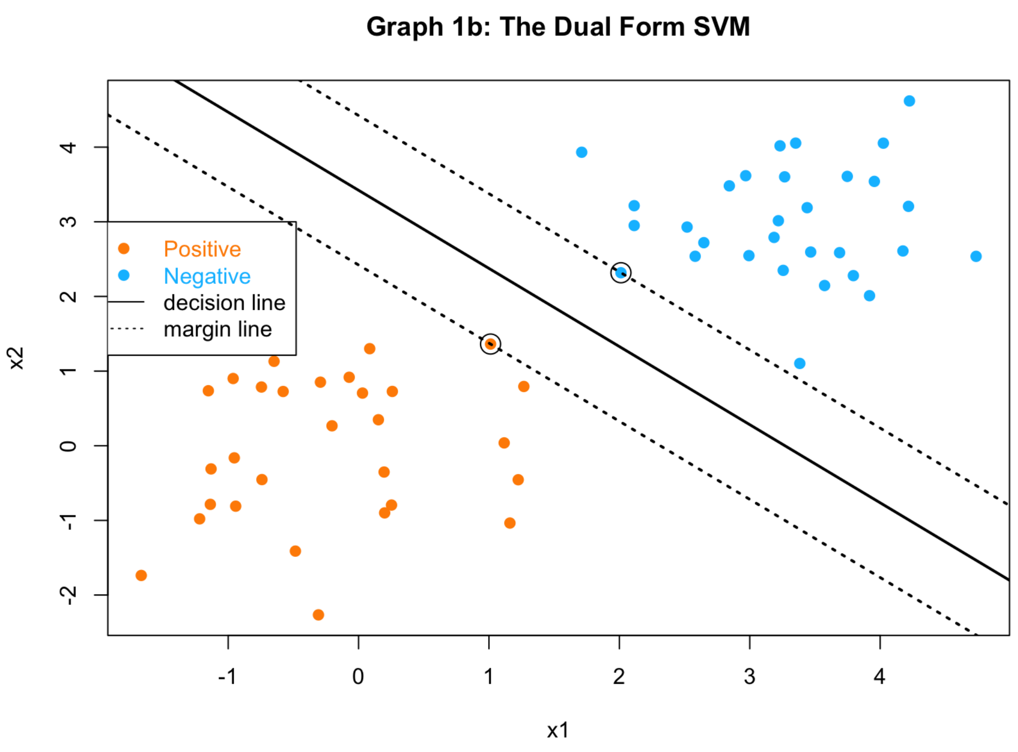
```r
library(tidyverse)

#create matrix D
prod <- sapply(1:nrow(x), function(i) y[i]*t(x)[,i])
Dmat2 <- t(prod)%*%prod

#create vector d
dvec2 = rep(1,nrow(x))

#create matrix A
Amat2 = matrix(0,nrow(x)+1,nrow(x))
Amat2[1,]=y
Amat2[2:(nrow(x)+1),]=diag(1,nrow(x),nrow(x))


#Add 10^{-10} to diagonal elements of D matrix
for(i in 1:nrow(x)) {
  Dmat2[i,i]=Dmat2[i,i]+10^{-10}
}

#create vector b
bvec2= rep(0,nrow(x)+1)

#find optimal alphas
alpha=solve.QP(Dmat2, dvec2, t(Amat2), bvec2, meq=1)$solution


#calculate product a*y*x
product=matrix(0,nrow(x),ncol(x))

for( i in 1: length(alpha)){

    product[i,1]=alpha[i]*y[i]*x[i,1]
    product[i,2]=alpha[i]*y[i]*x[i,2]

}

#calculate b1, b2 by summing a*y*x
beta2=rep(0,3)
beta2[2:3]=c(sum(product[,1]),sum(product[,2]))

#create 2 datsets, s.t y=1 for data1 and y=-1 for data_neg1
data1<-as.matrix(train %>% filter(y==1))
data_neg1<-as.matrix(train %>% filter(y==-1))

#calculate b0
beta2[1]=-1*(max(data_neg1[,1:2]%*%beta2[2:3])+min(data1[,1:2]%*%beta2[2:3]))/2
```

```
#create table of Betas from different methods
B0=c(round(b[1],3),round(beta2[1],3))
B1=c(round(b[2],3),round(beta2[2],3))
B2=c(round(b[3],3),round(beta2[3],3))
B<-matrix(c(B0, B1, B2), nrow = 3, byrow = TRUE)
Beta<-as.data.frame(B)
rownames(Beta)<-c("B0", "B1", "B2")
colnames(Beta)<-c("primal", "dual")

Beta
```

	primal <dbl>	**dual** <dbl>
B0	3.419	3.419
B1	-1.046	-1.046
B2	-0.999	-0.999

3 rows

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



#decision line
abline(a= -beta2[1]/beta2[3], b=-beta2[2]/beta2[3], col="black", lty=1, lwd = 2
)



#legend
op <- par(cex = 1)
    legend(-2,3, c("Positive","Negative","decision line", "margin line"), col=c
("darkorange", "deepskyblue", "black", "black"),
         pch=c(19, 19, NA, NA), lty=c(NA,NA,1,3),text.col=c("darkorange", "deep
skyblue", "black", "black"))

# mark the support vectors. The points were found by looking at graph and table
of x
points(x[28, 1], x[28, 2], col="black", cex=2)
points(x[49, 1], x[49, 2], col="black", cex=2)

# the two margin lines
abline(a= (-beta2[1]-1)/beta2[3], b=-beta2[2]/beta2[3], col="black", lty=3, lwd
= 2)
abline(a= (-beta2[1]+1)/beta2[3], b=-beta2[2]/beta2[3], col="black", lty=3, lwd
= 2)
title("Graph 1b: The Dual Form SVM")
```



```
#calculate and report ||Beta_1a-Beta_1b||
print(paste0("||B_Q1-B_Q2|| = ",sum(abs(b-beta2))))
```

```
## [1] "||B_Q1-B_Q2|| = 0.00017746780137351"
```

Question 2 [20 Points] Linearly nonseparable SVM

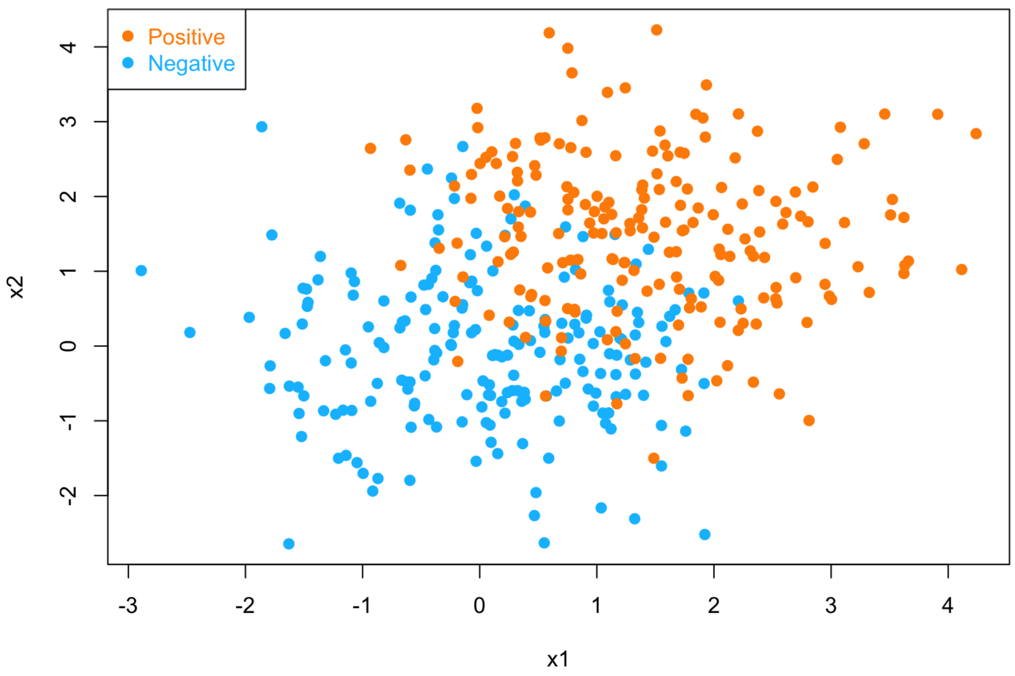
In this question, we will follow the formulation in Page 30 to solve a linearly nonseparable SVM. The dual problem is given by

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \ \text{ for } i = 1, \ldots, n \\ \text{and} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

Perform the following:

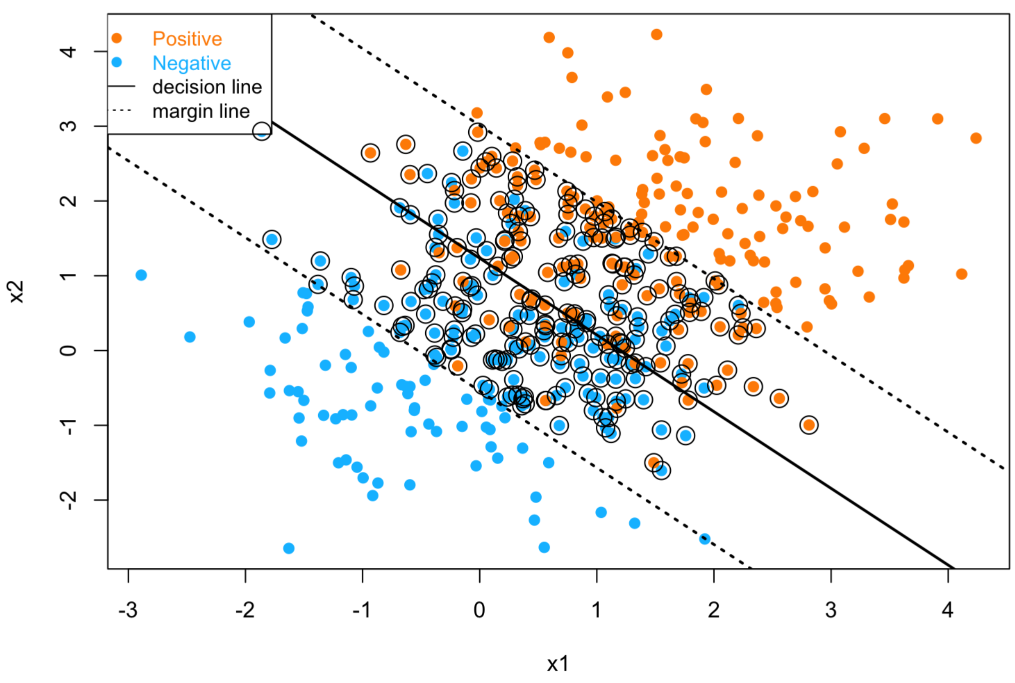
- Let $b = (\alpha_1, \ldots, \alpha_n)^T$. Then properly define $\mathbf{D}$, d, $\mathbf{A}$ and b_0 corresponding to this b for this problem. Use $C = 1$ as the penalty team.
- Note: Equality constrains can be addressed using the `meq` argument.
- Obtain the solution using the `solve.QP()` function, and convert the solution into $\boldsymbol{\beta}$ and β_0. Note:
  - use the information provided on page 32 to obtain the support vectors and β_0.
  - Your solution may encounter numerical errors, e.g., very small negative α values, or values very close to C. You could consider thresholding them to exactly 0 or C
  - Your $\mathbf{D}$ may not be definite positive, so consider adding 10^{-10} 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", y
lab = "x2")
  legend("topleft", c("Positive","Negative"), col=c("darkorange", "deepskyblue"
),
         pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```



```r
library(tidyverse)

#create matrix D
prod3 <- sapply(1:nrow(x), function(i) y[i]*t(x)[,i])
Dmat3 <- t(prod3)%*%prod3

#create vector d
dvec3 = rep(1,nrow(x))

#create matrix A
Amat3 = matrix(0,nrow(x)+1,nrow(x))
Amat3[1,]=y
Amat3[2:(nrow(x)+1),]=diag(1,nrow(x),nrow(x))

#Add 10^{-10} to diagonal elements of D matrix
for(i in 1:nrow(x)) {
  Dmat3[i,i]=Dmat3[i,i]+10^{-10}
}

#create vector b
C=1
bvec3= rep(C,nrow(x)+1)
bvec3[1]=0

#find alphas
alpha2=solve.QP(Dmat3, dvec3, t(Amat3), bvec3, meq=1)$solution

#calculate b1 abdd b2 by summing alpha*y*x
beta3=rep(0,3)
beta3[2]=sum(alpha2*y*x[,1])
beta3[3]=sum(alpha2*y*x[,2])

#create 2 datsets, s.t y=1 for data1 and y=-1 for data_neg1
data12<-as.matrix(train %>% filter(y==1))
data_neg12<-as.matrix(train %>% filter(y==-1))

#calculate b0
beta3[1]=-1*(max(data_neg12[,1:2]%*%beta3[2:3])+min(data12[,1:2]%*%beta3[2:3
]))/2


#find indexes of support vector points
index=c()
j=1
 for( i in 1:nrow(x)){
   if(x[i,2]>x[i,1]*(-beta3[2]/beta3[3])+(-beta3[1]+1)/beta3[3]||x[i,2]<x[i,1]*
      (-beta3[2]/beta3[3])+(-beta3[1]-1)/beta3[3]){
     index[j]=i
     j=j+1
     }
 }

f=c(1:nrow(x))
index2=f[!f %in% index]



#create plot
plot(x, col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", yla
b = "x2")

#decision line
abline(a= -beta3[1]/beta3[3], b=-beta3[2]/beta3[3], col="black", lty=1, lwd = 2
)



#legend
op <- par(cex = 0.9)
    legend(-3.3,4.5, c("Positive","Negative","decision line", "margin line"), c
ol=c("darkorange", "deepskyblue", "black", "black"),
         pch=c(19, 19, NA, NA), lty=c(NA,NA,1,3),text.col=c("darkorange", "deep
skyblue", "black", "black"))

# mark the support vectors.
for(i in 1:length(index2)){
  points(x[index2[i], 1], x[index2[i], 2], col="black", cex=2)

}



# the two margin lines
abline(a= (-beta3[1]-1)/beta3[3], b=-beta3[2]/beta3[3], col="black", lty=3, lwd
= 2)
abline(a= (-beta3[1]+1)/beta3[3], b=-beta3[2]/beta3[3], col="black", lty=3, lwd
= 2)
title("Graph 2: Linearly nonseparable SVM")
```

Graph 2: Linearly nonseparable SVM



```
#function that calculate predicted y using predicted beta and x
predict<-function(x, beta){

  f=apply(x, 1, function(x) beta[1]+t(x)%*%beta[2:3])
  y_pred=rep(0,length(f))

  for(i in 1:length(f)){
    if(f[i]>0){
      y_pred[i]=1
  }else{
    y_pred[i]=-1
  }
}
  y_pred
}



y_pred2=predict(x,beta3)

#calculate misclassification rate
misclass_rate2=as.numeric(table(y_pred2==y)[1])/length(y)

#report misclassification rate and betas
print(paste0("misclassification rate= ",misclass_rate2))
```

```
## [1] "misclassification rate= 0.1525"
```

```
print(paste0("b0 = ",beta3[1]))
```

```
## [1] "b0 = -0.695310643446315"
```

```
print(paste0("b1 = ",beta3[2]))
```

```
## [1] "b1 = 0.57762768259272"
```

```
print(paste0("b2 = ",beta3[3]))
```

```
## [1] "b2 = 0.562893155962229"
```

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:

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

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 (https://teazrq.github.io/stat542/other/r-intro.html) (Section 10).

We let $f(x)$ is to be a linear function, SVM can be solved by optimizing a penalized loss:

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

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.
- [10 pts] Write a penalized loss objective function `SVMfn(b, x, y, lambda)` and its gradient `SVMgn(b, x, y, lambda)`.
- [10 pts] Solve the coefficients using `optim()` and your objective and gradient functions with $\lambda = 1$ and `BFGS` method. Use 0 as the initialized value.

Report the followings:

- Your coefficients
- Your loss and mis-classification rate on training data.
- Plot all data and the decision line

- Hint: If you want to check your gradient function, you can run `optim()` without a this gradient function and compare the parameters to your previous ones. Note this will be much slower. You are not required to report this result.

3a

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

$$\frac{\partial h}{\partial \beta_0} = \sum_{i=1}^{n} \frac{\partial L}{\partial f} \frac{\partial f}{\partial \beta_0}$$
$$\frac{\partial h}{\partial \beta_0} = \sum_{i=1}^{n} \frac{1}{1+e^{-y_i f(x_i)}} * e^{-y_i f(x_i)} * -y_i$$
$$\frac{\partial h}{\partial \beta_0} = \sum_{i=1}^{n} \frac{-y_i e^{-y_i f(x_i)}}{1+e^{-y_i f(x_i)}}$$

$$\frac{\partial h}{\partial \beta} = \sum_{i=1}^{n} \frac{\partial L}{\partial f} \frac{\partial f}{\partial \beta} + 2\lambda\beta$$
$$\frac{\partial h}{\partial \beta} = \sum_{i=1}^{n} \frac{-y_i e^{-y_i f(x_i)}}{1+e^{-y_i f(x_i)}} * x_i^T + 2\lambda\beta$$
$$\frac{\partial h}{\partial \beta} = \sum_{i=1}^{n} \frac{-y_i x_i^T e^{-y_i f(x_i)}}{1+e^{-y_i f(x_i)}} + 2\lambda\beta$$

The final answers:

$$\frac{\partial h}{\partial \beta_0} = \sum_{i=1}^{n} \frac{-y_i e^{-y_i(\beta_0 + x_i^T \boldsymbol{\beta})}}{1+e^{-y_i(\beta_0 + x_i^T \boldsymbol{\beta})}}$$
$$\frac{\partial h}{\partial \beta} = \sum_{i=1}^{n} \frac{-y_i x_i^T e^{-y_i(\beta_0 + x_i^T \boldsymbol{\beta})}}{1+e^{-y_i(\beta_0 + x_i^T \boldsymbol{\beta})}} + 2\lambda\beta$$

3b and 3c

```r
#function that we need to minimize
SVMfn<-function(b, x, y, lambda){
  f<-apply( x, 1, function(x) b[1]+t(x)%*%b[2:length(b)])
  yf<- -y*f
  ln<-log(1+exp(yf))
  sum(ln)+lambda%*%t(b[2:length(b)])%*%b[2:length(b)]
}

#function that calculates gradient of B0, B1 ...Bn.
SVMgn<-function(b, x, y, lambda){

  f<-apply( x, 1, function(x) b[1]+t(x)%*%b[2:length(b)])
  yf<- -y*f
  num1<--y*exp(yf)
  denom1<-1+exp(yf)
  grad_B0=sum(num1/denom1)


  r=num1/denom1
  l=sapply(1:nrow(x), function(i) r[i]*t(x[i,]))

  sum_l=apply(l,1,sum)
  grad_B=sum_l+2*lambda%*%b[2:length(b)]
  sum_l=apply(l,1,sum)
  grad_B=sum_l+2*lambda%*%b[2:length(b)]

  grad=rep(0,length(b))
  grad[1]= grad_B0
  grad[2:length(b)]=grad_B
  grad
}

#loss function
loss<-function(b, x, y){
  f<-apply( x, 1, function(x) b[1]+t(x)%*%b[2:length(b)])
  yf<- -y*f
  ln<-log(1+exp(yf))
  sum(ln)
}

#find optimal betas
b=rep(0,ncol(x)+1)
b_optimal=optim(b, SVMfn, SVMgn, method = "BFGS", x = x, y = y, lambda=1)$par

#report betas, loss and misclassification rate
print(paste0("b0 = ",b_optimal[1]))
```

```
## [1] "b0 = -2.28022390135724"
```

```
print(paste0("b1 = ",b_optimal[2]))
```

```
## [1] "b1 = 1.57564948088927"
```

```
print(paste0("b2 = ",b_optimal[3]))
```

```
## [1] "b2 = 1.50980542451689"
```

```
print(paste0("loss = ",loss(b_optimal,x,y)))
```

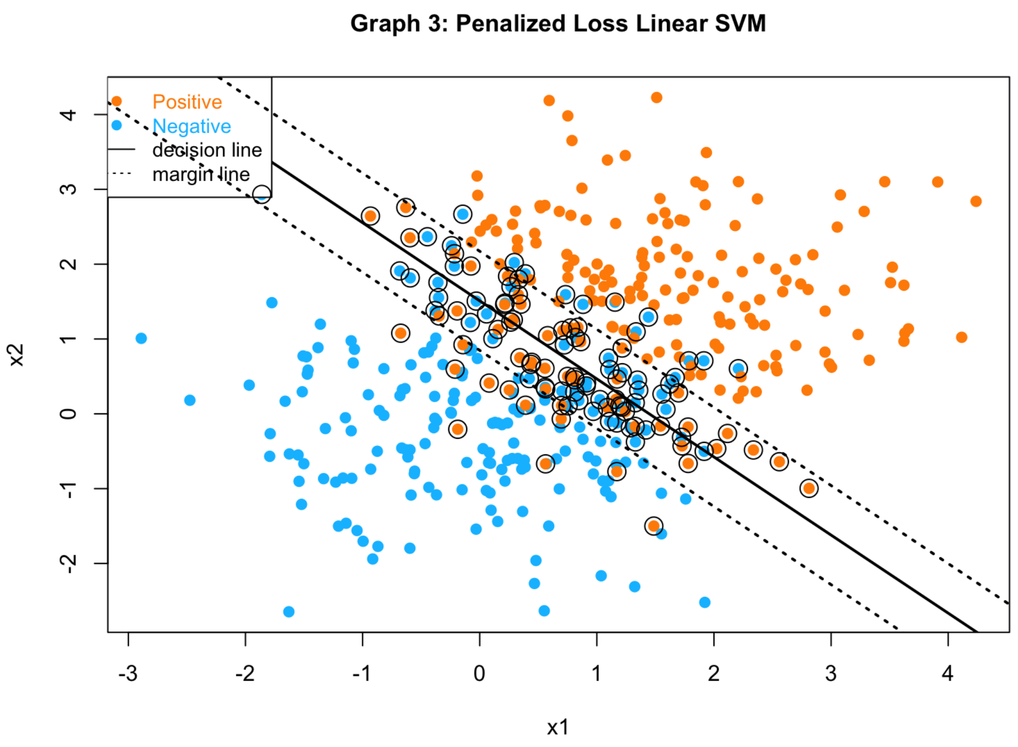
```
## [1] "loss = 125.097864960178"
```

```
y_pred3=predict(x,b_optimal)
misclass_rate3=as.numeric(table(y_pred3==y)[1])/length(y)
print(paste0("misclassification rate = ",misclass_rate3))
```

```
## [1] "misclassification rate = 0.1475"
```

```
#find indexes of support vector points
index=c()
j=1
 for( i in 1:nrow(x)){

   if(x[i,2]<x[i,1]*(-b_optimal[2]/b_optimal[3])+(-b_optimal[1]-1)/b_optimal[3]
&&
      y[i]==-1){
     index[j]=i
     j=j+1
   }

    if(x[i,2]>x[i,1]*(-b_optimal[2]/b_optimal[3])+(-b_optimal[1]+1)/b_optimal[3
]&&
       y[i]==1){
     index[j]=i
     j=j+1
   }
 }

f=c(1:nrow(x))

index2=f[!f %in% index]



#create plot
plot(x, col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", yla
b = "x2")

#decision line
abline(a= -b_optimal[1]/b_optimal[3], b=-b_optimal[2]/b_optimal[3], col="black"
, lty=1, lwd = 2)



#legend
op <- par(cex = 0.9)
    legend(-3.3,4.5, c("Positive","Negative","decision line", "margin line"), c
ol=c("darkorange", "deepskyblue", "black", "black"),
         pch=c(19, 19, NA, NA), lty=c(NA,NA,1,3),text.col=c("darkorange", "deep
skyblue", "black", "black"))

# mark the support vectors.
for(i in 1:length(index2)){
  points(x[index2[i], 1], x[index2[i], 2], col="black", cex=2)
  }



# the two margin lines
```

```
abline(a= (-b_optimal[1]-1)/b_optimal[3], b=-b_optimal[2]/b_optimal[3], col="bl
ack", lty=3, lwd = 2)
abline(a= (-b_optimal[1]+1)/b_optimal[3], b=-b_optimal[2]/b_optimal[3], col="bl
ack", lty=3, lwd = 2)
title("Graph 3: Penalized Loss Linear SVM")
```


Graph 3: Penalized Loss Linear SVM
Positive
Negative
decision line
margin line
x1
x2