STAT 542: Homework 6

Spring 2020, by Yifan Shi (yifans16)

Due: Monday, Apr 27 by 11:59 PM

Contents

Question 1 [40 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:

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

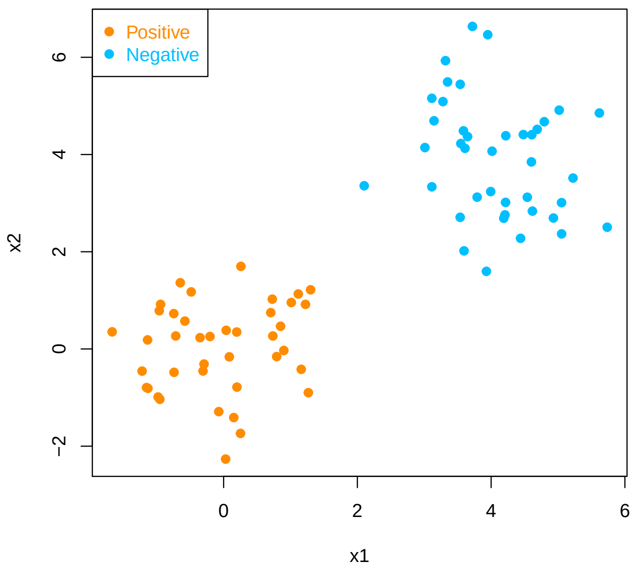
For more details, read the document file of the `quadprog` package on CRAN. You should generate the data using the following code (or write a similar code in Python). For each sub-question, perform the following:

- Propoerly define $\mathbf{D}$, d, A and a.
- Convert the solution into β and β_0, which can be used to define the classification rule
- Plot the decision line, the two margin lines and the support vectors

Note: The package requires $\mathbf{D}$ to be positive definite, while it may not be true in our case. 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 results slightly. So be careful when plotting your support vectors.

```
library(quadprog)
set.seed(3); 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)))

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"))
```

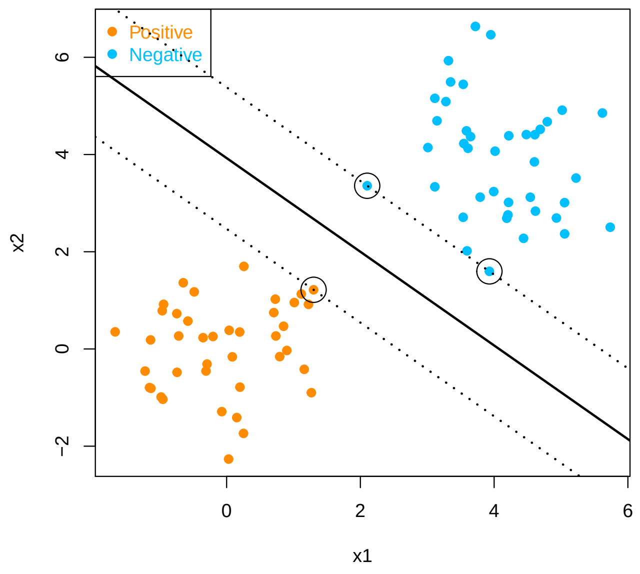


- [20 points] Formulate the SVM **primal** problem of the separable SVM formulation and obtain the solution to the above question.

Here I defined the β in the quadratic programing to be combination of $\hat{\beta}_0$ and $\hat{\beta}$. Then the matrix A is the product of each row in y and x (Here I added a column of 1's as the 1st column in x and call this matrix newx). Since $\hat{\beta} = Q\beta$, the matrix Q should be $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and matrix $D = Q^T Q$ should be $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. d is a vector of 0 in length of $p+1$ and a is a vector of 1 in length of $n * p$.

```
e1 = 1e-4
newx = cbind(matrix(1,n*p,1),x)
Qmat1 = matrix(0,2,3)
Qmat1[1,2] = 1; Qmat1[2,3] = 1
# matrices
Dmat1 = t(Qmat1)%*%Qmat1 + e1*diag(3)
Amat1 = sapply(1:(n*p), function(i) y[i]*t(newx)[,i])
# vectors
dvec1 = matrix(0,3,1)
avec1 = matrix(1,n*p,1)
# dual
prim = solve.QP(Dmat1,dvec1,Amat1,avec1,meq=0)
sol1 = matrix(prim$solution,3,1)
# SV, beta & beta0
sv1 = which(abs((newx%*%sol1))<=1+e1)
b_1 = Qmat1 %*% sol1
b0_1 = -sum(x[sv1[-3],]%*%b_1)/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[sv1, ], col="black", cex=3)
abline(a=-b0_1/b_1[2], b=-b_1[1]/b_1[2], col="black", lty=1, lwd = 2)
abline(a= (-b0_1-1)/b_1[2], b=-b_1[1]/b_1[2], col="black", lty=3, lwd = 2)
abline(a= (-b0_1+1)/b_1[2], b=-b_1[1]/b_1[2], col="black", lty=3, lwd = 2)
```

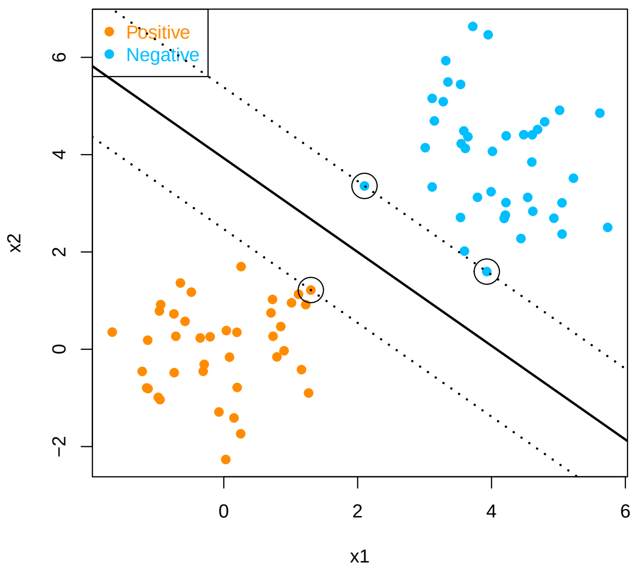


- [20 points] Formulate the SVM **dual** problem of the separable SVM formulation and obtain the solution to the above question.

Here I defined the β in the quadratic programing to be the vector α in length of $n * p$ and matrix Q to be 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 $n * p + 1$.

```
e2 = 1e-4
Qmat2 = sapply(1:(n*p), function(i) y[i]*t(x)[,i])
# matrices
Dmat2 = t(Qmat2)%*%Qmat2 + e2*diag(n*p)
Amat2 = t(rbind(matrix(y,1,n*p),diag(n*p)))
# vectors
dvec2 = matrix(1,n*p,1)
avec2 = matrix(0,n*p+1,1)
# dual
dual = solve.QP(Dmat2,dvec2,Amat2,avec2,meq=1)
sol2 = matrix(dual$solution,n*p,1)
# SV, beta & beta0
sv2 = which(abs(sol2)>=e2)
b_2 = t(sol2) %*% t(Qmat2)
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="black", lty=1, lwd = 2)
abline(a= (-b0_2-1)/b_2[2], b=-b_2[1]/b_2[2], col="black", lty=3, lwd = 2)
abline(a= (-b0_2+1)/b_2[2], b=-b_2[1]/b_2[2], col="black", lty=3, lwd = 2)
```



Question 2 [40 Points] Sovling SVM using Penalized Loss Optimization

Consider the 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 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 provided on our course website here (Section 10).

- [20 points] Linear SVM. When $f(x)$ is a linear function, SVM can be solved by optimizing the penalized loss:

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

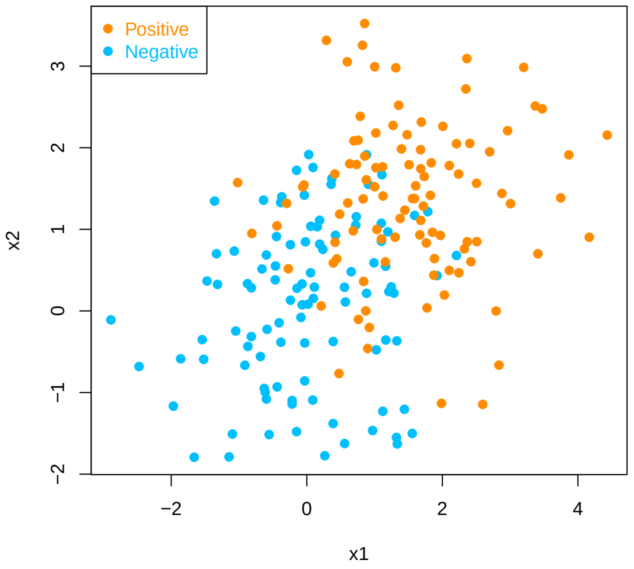
  You should generate the data using the code provided below code (or write similar code in Python), and answer these questions:
  - Write a function to define the penalized loss objective function. The algorithm can run faster if you further define the gradient function. Hence, you should also define 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.

```r
set.seed(20)
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")
  legend("topleft", c("Positive","Negative"), col=c("darkorange", "deepskyblue"),
         pch=c(19, 19), text.col=c("darkorange", "deepskyblue"))
```



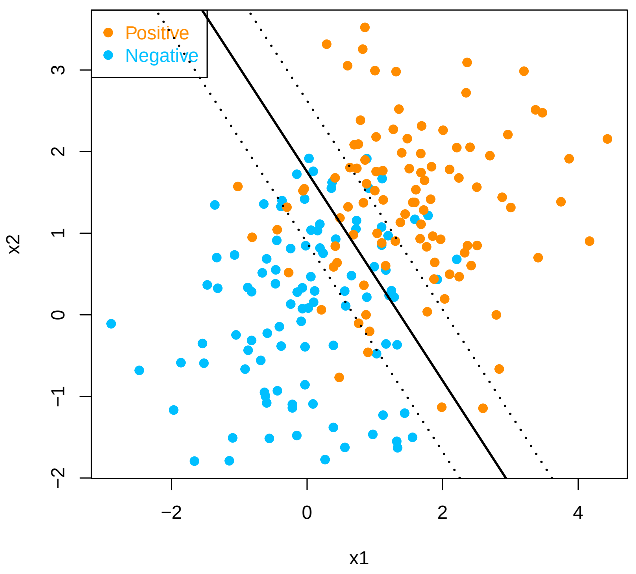
```
x1 = cbind(matrix(1,n*p,1),x)
# loss function
loss1 <- function(b){
  b = matrix(b)
  t1 = sum(log(1+exp(-y*(x1%*%b))))
  t2 = 0.1*t(b[-1,])%*%b[-1,]
  return(t1+t2)
}
# gradient function
grad1 <- function(b){
  b = matrix(b)
  g11 = -1/(1+exp(-y*(x1%*%b)))*exp(-y*(x1%*%b))
  g12 = sapply(1:(n*p),function(i) y[i]*matrix(x1[i,]))
  g13 = apply(sapply(1:(n*p),function(i) g12[,i]*g11[i]),1,sum)
  g2 = 2*0.1*matrix(c(0,b[-1,]))
  return(matrix(g13)+g2)
}
# optimization
opt1 = optim(rep(0,3),fn=loss1,gr=grad1,method="BFGS")
beta0_1 = opt1$par[1]
beta_1 = opt1$par[-1]
beta_1
```

```
## [1] 1.469999 1.148739
```

```
beta0_1
```

```
## [1] -2.013287
```

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



- [20 points] Non-linear SVM. You should generate the data using the code provided below code (or write similar code in Python), and answer these questions:
  - Use the kernel trick to solve for a nonlinear decision rule. The penalized optimization becomes

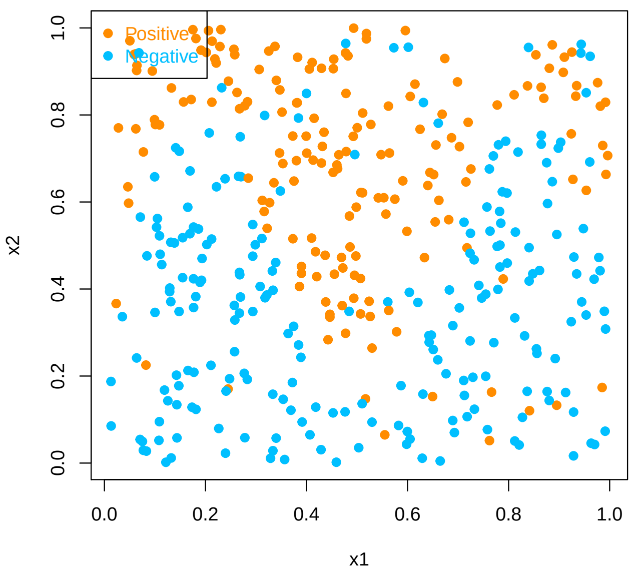
$$\sum_{i=1}^{n} L(y_i, K_i^T \beta) + \lambda \beta^T K \beta$$

    where K_i is the ith column of the $n \times n$ kernel matrix K. For this problem, we consider the Gaussian kernel. For the bandwidth of the kernel function, you can borrow ideas from our previous homework. You do not have to tune the bandwidth.
  - Pre-calculate the $n \times n$ kernel matrix K of the observed data.
  - Write a function to define the objective function. 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.
  - 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. 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. You can judge if your model is reasonable based on the true model.

```
set.seed(1); n = 400; p = 2

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

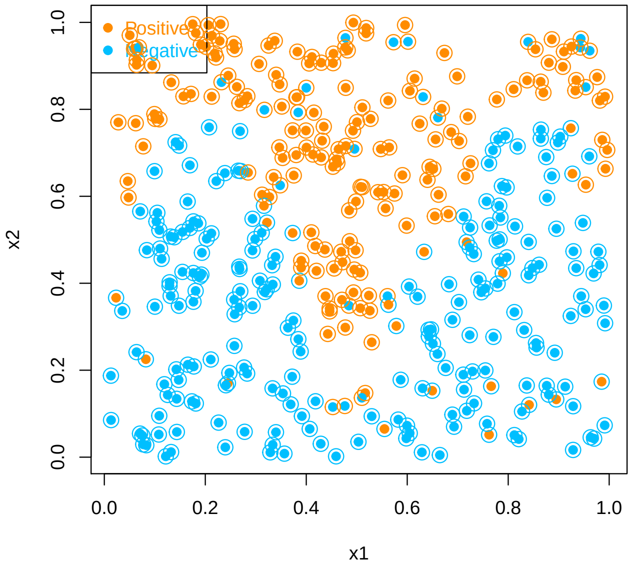


```r
# find the kernel matrix
K.Gauss <- function(u){
  1 / sqrt(2*pi) * exp(-(u^2)/2)
}
lambda = apply(x,2,sd)*(4/3/nrow(x))^(1/5)
K.fun <- function(u) {K.Gauss(u/lambda)/lambda}
K = matrix(NA,n,n)
for (i in 1:n){
  K[,i] = apply(sapply(1:n, function(j) K.fun(x[i,]-x[j,])),2,mean)
}
```

```r
# loss function
loss2 <- 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
grad2 <- function(b){
  b = matrix(b)
  g11 = -1/(1+exp(-y*(K%*%b)))*exp(-y*(K%*%b))
  g12 = sapply(1:n,function(i) y[i]*matrix(K[i,]))
  g13 = apply(sapply(1:n,function(i) g12[,i]*g11[i]),1,sum)
  g2 = 2*0.5*matrix(c(0,b[-1,]))
  return(matrix(g13)+g2)
}
# optimization
opt2 = optim(rep(0,n),fn=loss2,gr=grad2,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: 0.1075"
```

```r
plot(x,col=ifelse(y>0,"darkorange", "deepskyblue"), pch = 19, xlab = "x1", ylab = "x2",cex=1)
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)
```



Question 3 [20 Points] Gradient Boosting using `xgboost`

We will use the handwritten digit recognition data again from the `ElemStatLearn` package. We only consider the train-test split, with the pre-defined `zip.train` and `zip.test`. However, **use `zip.test` as the training data, and `zip.train` as the testing data!** For this question, we again only consider the two digits: 2 and 4. We will use cross-validation to choose the best tuning. For this question, use the `xgboost` package (available in both R and Python), and consider two tuning parameters when you perform cross-validation (using their built-in functions): the base learner (linear or tree), tree depth (for trees only) and the learning rate. Choose a reasonable range of tunings and use the area under the ROC curve (AUC) as the criteria to select the best tuning. Report the prediction accuracy of your final model.

```r
  # Handwritten Digit Recognition Data
  library(ElemStatLearn)
  library(xgboost)

  # this is the training data!
  dim(zip.test)
```

```
## [1] 2007  257
```

```r
  train = zip.test

  # this is the testing data!
  dim(zip.train)
```

```
## [1] 7291  257
```

```r
  test = zip.train

  #subset
  trn = train[which(train[,1]==2|train[,1]==4),]
  tst = test[which(test[,1]==2|test[,1]==4),]
  #set the label to be either 0 or 1
  trn[,1] = ifelse(trn[,1]==2,0,1)
  tst[,1] = ifelse(tst[,1]==2,0,1)
```

Cross validation for linear:

```
set.seed(542)
dtrn <- xgb.DMatrix(trn[,-1], label = trn[,1])
param1 = list(booster="gblinear",objective="reg:logistic")
bst1 = xgb.cv(data=dtrn,nrounds=30,params=param1,verbose=0,
              nfold=10,metrics="auc",early_stopping_rounds=3)
```

Cross validation for tree:

```
set.seed(542)
lr = seq(0.02,0.3,0.04)
td = seq(3,10,1)
auc = matrix(NA,length(lr),length(td))
itr = matrix(NA,length(lr),length(td))
for (l in 1:length(lr)){
  for (d in 1:length(td)){
    param2 = list(booster="gbtree",eta=lr[l],max_depth=td[d])
    bst2 = xgb.cv(data=dtrn,nrounds=30,params=param2,verbose=0,
                  nfold=10,metrics="auc",early_stopping_rounds=3)
    auc[l,d] = max(bst2[["evaluation_log"]][["test_auc_mean"]])
    itr[l,d] = which.max(bst2[["evaluation_log"]][["test_auc_mean"]])
  }
}
```

Find the best tuning parameters:

```
print(paste0("iterations for tree: ", itr[which.max(auc)]))
```

```
## [1] "iterations for tree: 22"
```

```
print(paste0("lr for tree: ", lr[which(auc==max(auc),arr.ind=T)[1]]))
```

```
## [1] "lr for tree: 0.22"
```

```
print(paste0("max depth for tree: ", td[which(auc==max(auc),arr.ind=T)[2]]))
```

```
## [1] "max depth for tree: 3"
```

```
print(paste0("iterations for linear: ", which.max(bst1[["evaluation_log"]][["test_auc_mean"]])))
```

```
## [1] "iterations for linear: 7"
```

I decided to compare the prediction accuracy on the test set of linear and tree with their best tuning parameters.

```
set.seed(542)
xgbst1 = xgb.train(data=dtrn,nrounds=7,params=param1,verbose=0)
param2 = list(booster="gbtree",eta=0.22,max_depth=3,num_class=2)
xgbst2 = xgb.train(data=dtrn,nrounds=22,params=param2,verbose=0)
```

```
pred1 = predict(xgbst1,tst[,-1])
pred1 = ifelse(pred1>=0.5,1,0)
print(paste0("accuracy for linear: ", mean(pred1==tst[,1])))
```

```
## [1] "accuracy for linear: 0.963123644251627"
```

```r
pred2 = predict(xgbst2,tst[,-1])
print(paste0("accuracy for tree: ", mean(pred2==tst[,1])))
```

```
## [1] "accuracy for tree: 0.971800433839479"
```

Thus 22 iterations of the tree model with learning rate 0.22 and maximum depth of 3 gives me the best prediction accuracy.