STAT 6390 - Deep Learning - Mini Project 1

Nisansala Wickramasinghe

Question 1

a) **Figure** 1 shows plot of training data. Data seems linearly separable. If we take 45° line as the linear decision boundary we do not see overlapping red and blue points near the boundary. Decision boundary is plotted in **Figure** 1

b) R code for the implementation of perceptron can be found in section 2. Percepron criterion was used as the loss function. For a misclassified point, the gradient of loss function is $-y\overrightarrow{x} = -\{(y-\hat{y})/2\}\overrightarrow{x}$. Then the gradient-descent update becomes $\overrightarrow{w} \leftarrow \overrightarrow{w} + \alpha(y-\hat{y})\overrightarrow{x}$. Algorithm converge at 4th epoch (see **Figure** 1).

Training error : 0

Test error : 0.005

c) R code for the implementation of perceptron can be found in section 2. Hinge loss was used as the loss function. For a misclassified point, the gradient of hinge loss is $-y\overrightarrow{x} = -\{(y-\hat{y})/2\}\overrightarrow{x}$ which is the same as the grdient of perceptron criterion. Then the gradient-descent update becomes $\overrightarrow{w} \leftarrow \overrightarrow{w} + \alpha(y-\hat{y})\overrightarrow{x}$. Therefore results are similar to part b) and algorithm converge at 4th epoch (see **Figure** 1).

Training error : 0

Test error : 0.005

d) Decision boundaries of classifiers obtained in parts b) and c) are plotted in **Figure** 1. Decision boundries for part b) and c) are quite similar with my guess.

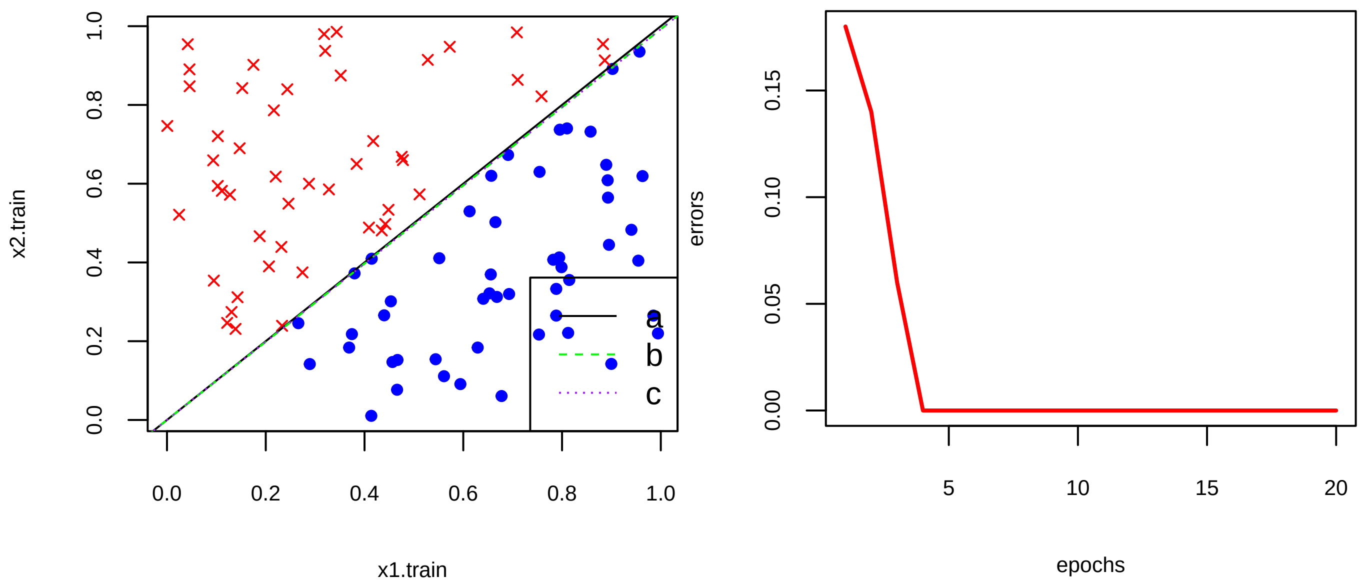

Figure 1: *Left: Decision boundaries for part a),b),c) Right: training error vs epochs*

e) Both classifiers, classifier with perceptron criterion and classifier with hinge loss, gives the same test error for the data provided as they have the same gradient and weight update rule is the same. Data are linearly separable. Therefore both classifiers performs well with 0.005 test error rate.

Question 2

a) The scatterplots and correlation matrix (see **Figure** 2) show that there is a positive correlation between response variable `Sales` and predictor variables `TV` and `radio`. The points in Q-Q plot deviate from the straight line suggesting that normality assumption is violated.

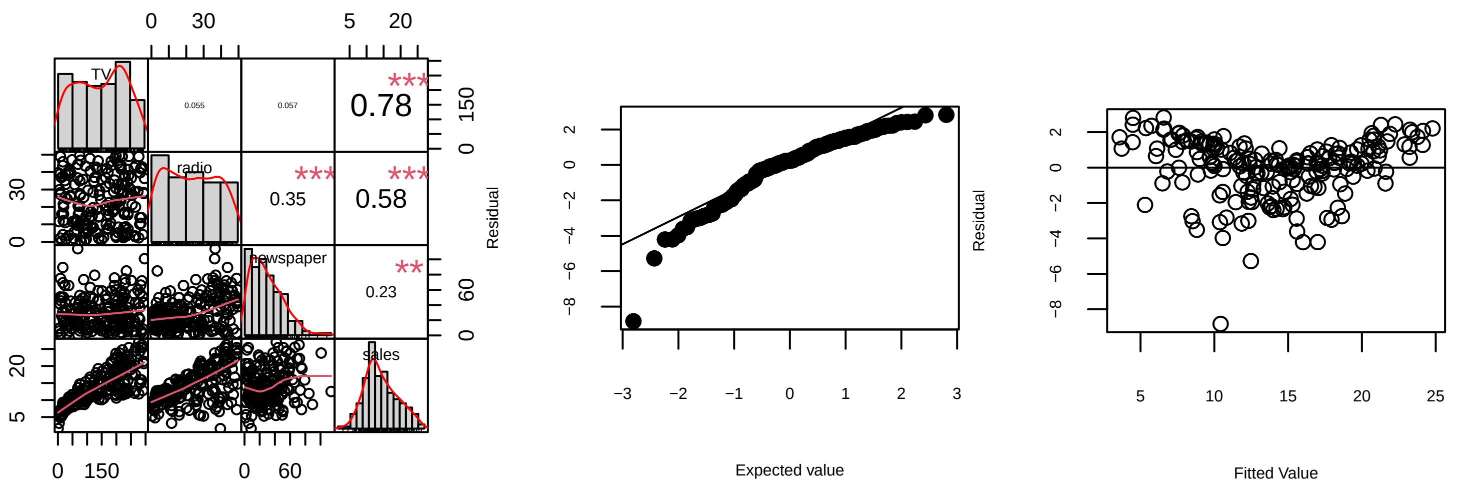

Figure 2: *Exploratory analysis. Left:Scatterplot matrix for Advertising data. Middle:Q-Q plot, Right:Residual vs fitted value*

b) Summary statistics for the model with all variables given below. We **normalized X** as they are in different scales. Based on the summary, we reject the null hypothesis $H0 : \beta_j = 0$ for the predictor `TV` and ‘radio“. The results are consistent with what we found in the exploratory analysis in part (a).

The least squares method is used for developing estimates. Consider linear regression model $\mathbf{Y} = \mathbf{X}\beta + \epsilon$. Then fitted values will be $\hat{\mathbf{Y}} = \mathbf{X}\beta$. We find least square estimates $\hat{\beta}$ by minimizing the sum of squares of residuals. $\epsilon'\epsilon = (\mathbf{Y} - \mathbf{X}\beta)'(\mathbf{Y} - \mathbf{X}\beta)$. To minimize $\epsilon'\epsilon$ w.r.t β, solve the normal equations $\partial\epsilon'\epsilon/\partial\beta = -2\mathbf{X}'\mathbf{Y} + 2\mathbf{X}'\mathbf{X}\beta = 0$ and we get least square estimates $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$.

```
Call:
lm(formula = y ~ TV + radio + newspaper, data = norm_Advertising)

Residuals:
    Min      1Q  Median      3Q     Max 
-8.8277 -0.8908  0.2418  1.1893  2.8292 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  14.0225     0.1192 117.655   <2e-16 ***
TV            3.9291     0.1198  32.809   <2e-16 ***
radio         2.7991     0.1278  21.893   <2e-16 ***
newspaper    -0.0226     0.1279  -0.177     0.86    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8956 
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

c) R code for the implementation of gradient-descent algorithm can be found in section 2. Algorithm converge when number of iterations is 1000 and learning rate 0.001. The estimates obtained in part(b) are similar to the estimates obtained in part(c).

```
[1] "coefficients"

                 [,1]
Intercept 14.02250000
TV         3.92908869
radio      2.79906919
newspaper -0.02259517
```

Section 2: R code

```r
## ----include=FALSE-----------------------------------------------------------------
#Data preprocessing
train.data = read.csv("1-training-data.csv", header = TRUE)
str(train.data)
train.X = subset(train.data, select = c("x1.train", "x2.train"))
train.Y = train.data$y.train
test.data = read.csv("1-test-data.csv", header = TRUE)
str(test.data)
test.X = subset(test.data, select = c("x1.test", "x2.test"))
test.Y = test.data$y.test


## ----include=FALSE-----------------------------------------------------------------

######## 1a) ###########
# Can be found in 1d)

######## 1b) Percepron criterion ###########

# define activation function
sign_func <- function(z){
  return(if (z < 0) -1 else 1)
}

# function to calculate derivative of perceptron loss
perceptron_loss_prime <- function(y, yhat, x){
  return(- (y - yhat) * x)
}

perceptron <- function(X, Y, l_rate, epochs, max_iter){
  # Insering 1 for bias, x0 = 1 to the dataframe.
  X <- data.frame(x0 = 1, X)

  # Initializing weights to zeros
  weights  <- rep(0, dim(X)[2])

  # Initializing errors vector
  errors <- rep(0, epochs)

  for(j in 1: epochs){
    for(i in 1:length(Y)){
      x <- as.numeric(X[i,])
      y <- Y[i]

      # Forward propagation and calculating prediction
      yhat = sign_func(weights %*% x)

      #Calculate errors using perceptron criterion
      if (max(-y * yhat, 0) != 0.0) {
        errors[j] <- errors[j] + 1
      }

      #Updating weight
      deltaL = perceptron_loss_prime(y, yhat, x)
      weights <- weights - l_rate * deltaL
    }
    #stopping criterion
    if (epochs > max_iter){
      print("exceeds maximum number of iteration")
      break
    }
    #print(paste("Epoch", j,"/",epochs,"| loss", errors[j]/length(Y)))
  }
  return(list(weights = weights, errors = errors/length(Y)))
}

epochs = 20
l_rate = 1

out = perceptron(train.X,train.Y, l_rate, epochs, 100)
#print("Training data: \n")
#out

#plot(1:epochs, err, type="l", lwd=2, col="red", xlab="epochs", ylab="errors")


## ----include=FALSE------------------------------------------------------------
# make prediction
y_all = as.matrix(data.frame(x0 = 1, test.X)) %*% as.matrix(out$weights)

y_pred = y_all
y_pred[y_all >= 0] = 1
y_pred[y_all< 0] = -1

err = sum(y_pred != test.Y)/length(test.Y)
#print("Test data: \n")
#print(err)


## ----include=FALSE------------------------------------------------------------

######## 1c) Using Hinge loss ###########

# define activation function
sign_func <- function(z){
  return(if (z < 0) -1 else 1)
}

# function to calculate derivative of perceptron loss
hinge_loss_prime <- function(y, yhat, x){
  return(- (y - yhat) * x)
}

perceptron <- function(X, Y, l_rate, epochs, max_iter){
  # Insering 1 for bias, x0 = 1 to the dataframe.
  X <- data.frame(x0 = 1, X)

  # Initializing weights to zeros
  weights  <- rep(0, dim(X)[2])

  # Initializing errors vector
  errors <- rep(0, epochs)

  for(j in 1: epochs){
    for(i in 1:length(Y)){
      x <- as.numeric(X[i,])
      y <- Y[i]
```

```r
      # Forward propagation and calculating prediction
      yhat = sign_func(weights %*% x)

      #Calculate errors using hinge loss
      if (max(1 - y * yhat, 0) != 0.0) {
        errors[j] <- errors[j] + 1
      }

      #Updating weight
      deltaL = hinge_loss_prime(y, yhat, x)
      weights <- weights - l_rate * deltaL
    }
    #stopping criterion
    if (epochs > max_iter){
      print("exceeds maximum number of iteration")
      break
    }
    #print(paste("Epoch", j,"/",epochs,"| loss", errors[j]/length(Y)))
  }
  return(list(weights = weights, errors = errors/length(Y)))
}

epochs = 20
l_rate = 1

out_hinge = perceptron(train.X,train.Y, l_rate, epochs, 100)
#print("Training data: \n")
#out_hinge


## ----include=FALSE----------------------------------------------------------------
# make prediction
y_all = as.matrix(data.frame(x0 = 1, test.X)) %*% as.matrix(out_hinge$weights)

y_pred = y_all
y_pred[y_all >= 0] = 1
y_pred[y_all< 0] = -1

err = sum(y_pred != test.Y)/length(test.Y)
#print("Test data: \n")
#print(err)


## ----q1bc, eval=FALSE, fig.height=3, fig.width=6, include=FALSE------------------
## par(mfrow=c(1,2))
## plot(1:epochs, out$errors, type="l", lwd=2, col="red", xlab="epochs", ylab="errors")
## plot(1:epochs, out_hinge$errors, type="l", lwd=2, col="red", xlab="epochs", ylab="errors")

######## 1d) Plotting Decision boundaries ###########

## ----q1d, echo=FALSE, fig.cap="\\textit{Plot for training data}",fig.height=4, fig.width=5----

# Save current graphical parameters
opar <- par(no.readonly = TRUE)

# Change the margins of the plot
# (the fourth is the right margin)
par(mar = c(6, 5, 4, 6))
```

```r
#Drawing Decision boundary
plot(train.X, pch = ifelse(train.Y == 1, 19, 4),
     col = ifelse(train.Y == 1 , "blue", "red"), cex = 0.7, cex.lab=0.7, cex.axis=0.7)
abline(0,1, col="black", lwd=1, lty=1)
abline(a = -1.0*out$weights[1]/out$weights[3], b = -1.0*out$weights[2]/out$weights[3], col="green", lwd=1, lty=2
abline(a = -1.0*out_hinge$weights[1]/out_hinge$weights[3], b = -1.0*out_hinge$weights[2]/out_hinge$weights[3], c
legend(x = "topright",
       inset = c(-0.35, 0),
       legend = c("a", "b", "c"),
       lty = c(1, 2, 3),
       col = c("black","green","purple"),
       lwd = 2,
       xpd = TRUE) # Needed to put the legend outside the plot

# Back to the default graphical parameters
on.exit(par(opar))

########################### Question 2 ############################################

######## 2a) exploratory data analysis ###########

## ----include=FALSE---------------------------------------------------------------
Advertising <- read.csv("Advertising.csv", header = TRUE)
Advertising <- Advertising[,-1]
full.model <- lm(sales~ TV + radio + newspaper , data = Advertising)

## ----q2a, echo=FALSE, fig.cap="\\textit{Exploratory analysis. Left:Scatterplot matrix for Advertising data. Mi
chart.Correlation(Advertising)

qqnorm(full.model$residuals, main = " ", xlab = "Expected value", ylab = "Residual",pch = 19,cex.lab=0.5,xaxt="n
qqline(full.model$residuals)
axis(2,cex.axis=0.5)
axis(1,cex.axis=0.5)

plot(x = full.model$fitted.values,  main = " ", y = full.model$residuals, abline(0,0), xlab = "Fitted Value",yla
axis(2,cex.axis=0.5)
axis(1,cex.axis=0.5)

######## 2b) Fit linear regression model ###########

## ----include=FALSE---------------------------------------------------------------
X = Advertising[,1:3]
y = Advertising[,4]

#normalize data
X = apply(X, 2, function(x) (x-mean(x))/sd(x))
norm_Advertising <- as.data.frame(cbind(X,y))

## ----include=FALSE---------------------------------------------------------------
library(PerformanceAnalytics)
full.model <- lm(y ~ TV + radio + newspaper , data = norm_Advertising)


## ----echo=FALSE------------------------------------------------------------------
summary(full.model)

######## 2c) Gradient descent algorithm ###########

## ----echo=FALSE------------------------------------------------------------------
```

```r
Gradient_descent <- function(X, y, l_rate, n_iter, max_iter){
  X <- as.matrix(data.frame(Intercept = 1, X))

  weights  <- matrix(rep(0, dim(X)[2]), ncol=1)

  MSE <- rep(0, n_iter)

  for (i in 1:n_iter) {
    yhat <- X %*% weights
    MSE[i] <- sum( (y - yhat)^2 ) / length(y)
    deltaL = -t(y-yhat) %*% X
    weights <- weights - l_rate * t(deltaL)
    if (n_iter > max_iter){
      print("exceeds maximum number of iteration")
      break
    }
  }
  return(list(weights = weights, errors = MSE[n_iter]))
}

out = Gradient_descent(X,y, 0.001, 1000,10000)
print("coefficients")
out$weights
```