HW2

Kendra Chalkley

May 3, 2018

```
## -- Attaching packages ---------------------------------------------------------------------------

## √ ggplot2 2.2.1     √ purrr   0.2.4
## √ tibble  1.4.2     √ dplyr   0.7.4
## √ tidyr   0.8.0     √ stringr 1.3.0
## √ readr   1.1.1     √ forcats 0.3.0

## -- Conflicts ---------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

Problem 1

1.1 Explanatory data analysis

a) How many binary attributes are in the dataset? List them

There is ony one binary attribute: column 4: chas

b) Correlations in between the first 13 attributes and the target attribute

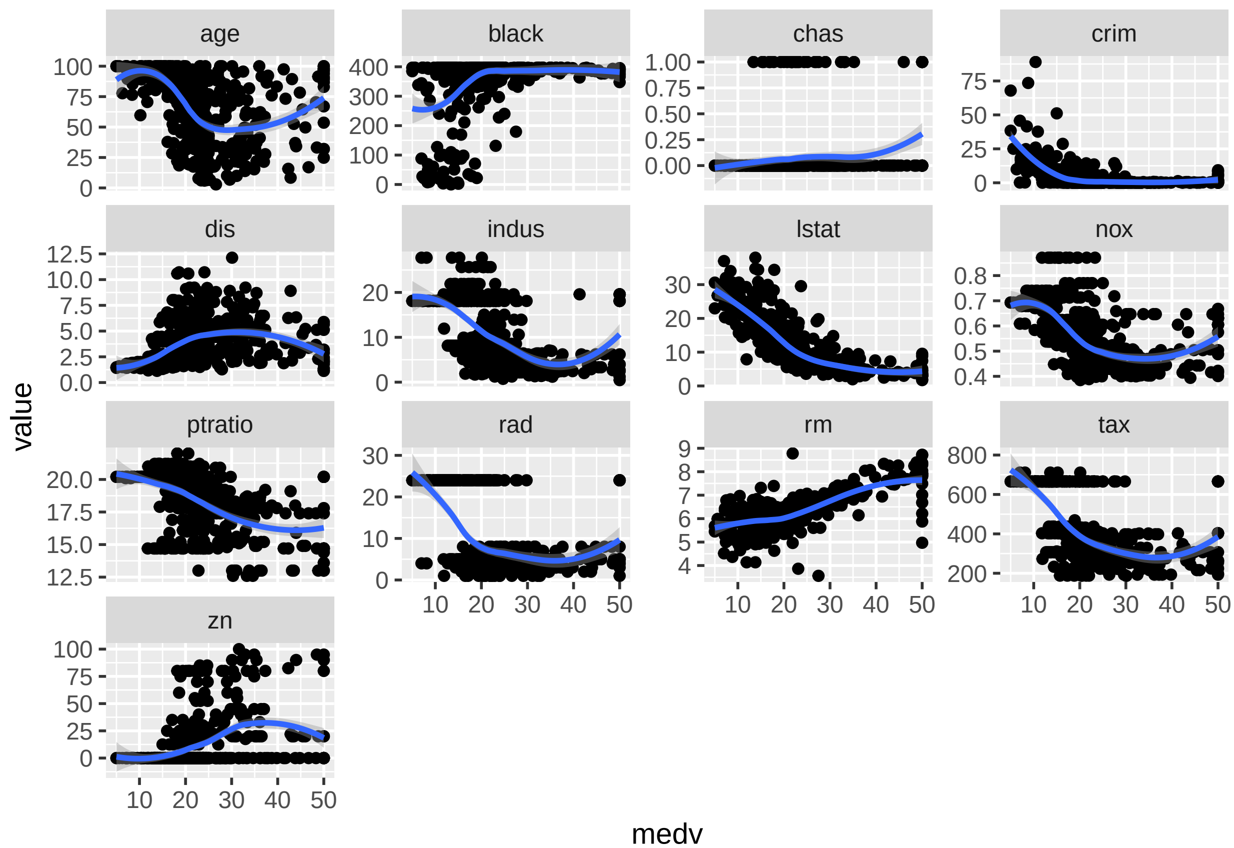
What are the attribute names with the highest positive and negative correlations to the target attribute?

```
##       crim         zn      indus       chas        nox         rm
## -0.3883046  0.3604453 -0.4837252  0.1752602 -0.4273208  0.6953599
##        age        dis        rad        tax    ptratio      black
## -0.3769546  0.2499287 -0.3816262 -0.4685359 -0.5077867  0.3334608
##      lstat       medv
## -0.7376627  1.0000000
```

The highest positive correlation is in column 6 (rm) with a value of 69.5% and the highest negative correlation is in column 13 (lstat) with a value of -73.8%

c) Scatter plots

```
## `geom_smooth()` using method = 'loess'
```



rm (column 6) looks most linear. the binary value, chas (column 4) seems least linear, not only because it's binary, but the 1 values are in the center of the plot with the 0s at either end.

d) Full correlation matrix

The highest correlation is between rm and tax (columns 6 and 7) at 91%

1.2 Linear Regression

a) LR_solve

The funciton below uses R's built in linear regression to find coefficients. The more appropriate calculation would be: `w <- ginv(t(X)%*%X) %*% t(X) %*%y` where `t(X)` indicates the X^T, `ginv(X)` indicates the X^{-1}, and `X%*%y` indicates Xy. However, this is just the very beginning of my matrix multiplication problems which I have been unable to solve in many many hours of trying, so I've used R's linear model to obtain these coefficients as a solid first step.

```
LR_solve <- function(X,y){
  model<- lm(y~X[,1]+
        X[,2]+
```

```
       X[,3]+
       X[,4]+
       X[,5]+
       X[,6]+
       X[,7]+
       X[,8]+
       X[,9]+
       X[,10]+
       X[,11]+
       X[,12]+
       X[,13]
     )
#

    w <- summary(model)$coefficients[,'Estimate']
  return(w)
}
```

b) LR_predict

```
LR_predict <- function(X,w){
  X <- as.matrix(add_column(X, intercept=1, .before=1))

  num_obs <-nrow(X)
  num_feature <- ncol(X)
  predicts <- integer()

  for (i in 1:num_obs){
#    print(w)
 #   print(length(w))
  #  print(X[i,])
  #  print(length(X[i,]))
    predicts[i] <- w%*%X[i,]
  }
  return(predicts)
}
```

c) main3_2

First a formula to calculate mean squared error:

```
MSE_calc <- function(predictions, reality){
  mse <- 0
  totalerror <- 0

  num_predictions <-length(predictions)
  for( i in 1:num_predictions){
    dif <- predictions[i]-reality[i]

    sqdif <- dif^2
```

```
    totalerror <- totalerror+sqdif
  }

  mse <- (totalerror/num_predictions)
  return(mse)
}
```

And then the actual calculation of values:

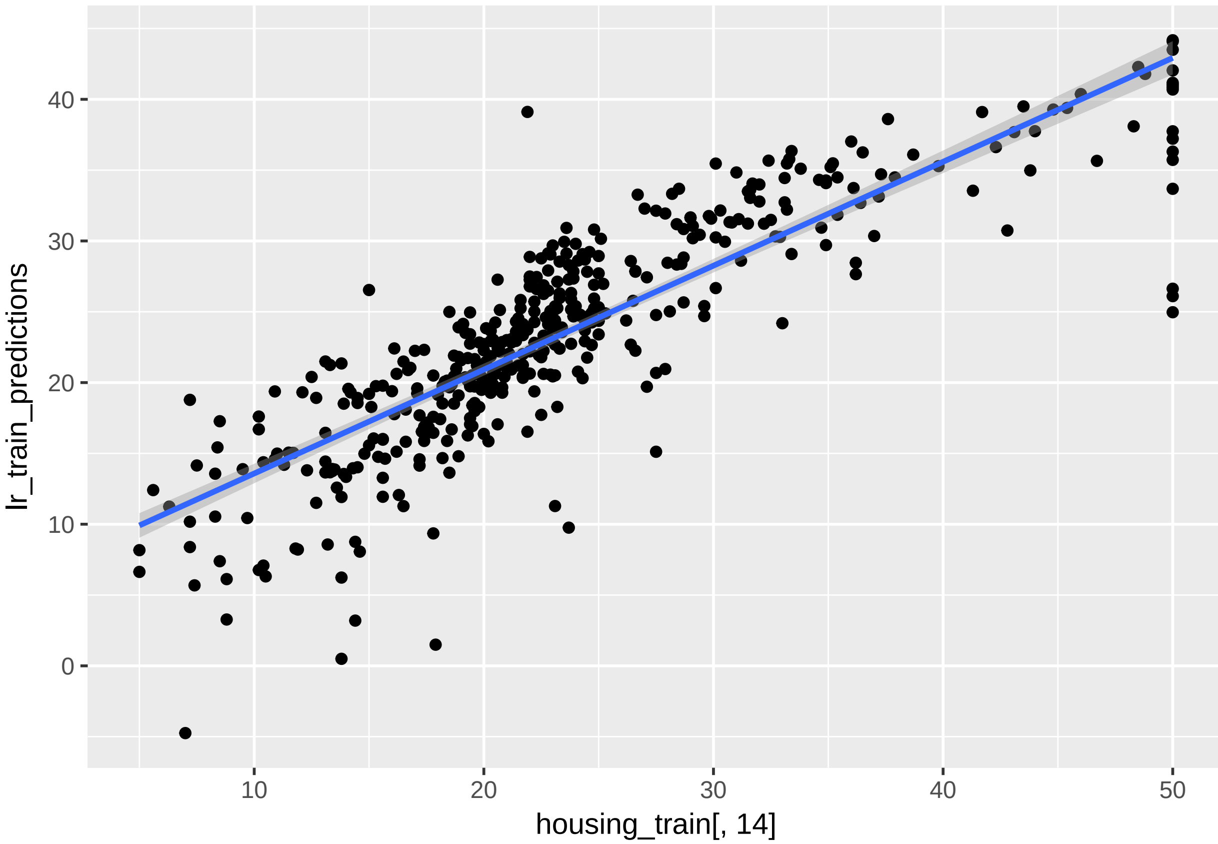
```
housing_train <- housing[1:433,]
housing_test <- housing[434:506,]

lr_weights <- LR_solve(data.frame(housing_train[,1:13]),housing_train[,14])

lr_train_predictions <- LR_predict(housing_train[,1:13],lr_weights)
lr_test_predictions <- LR_predict(housing_test[,1:13],lr_weights)

trainerror <- MSE_calc(lr_train_predictions,housing_train[,14])
testerror <- MSE_calc(lr_test_predictions,housing_test[,14])
```

d) Report

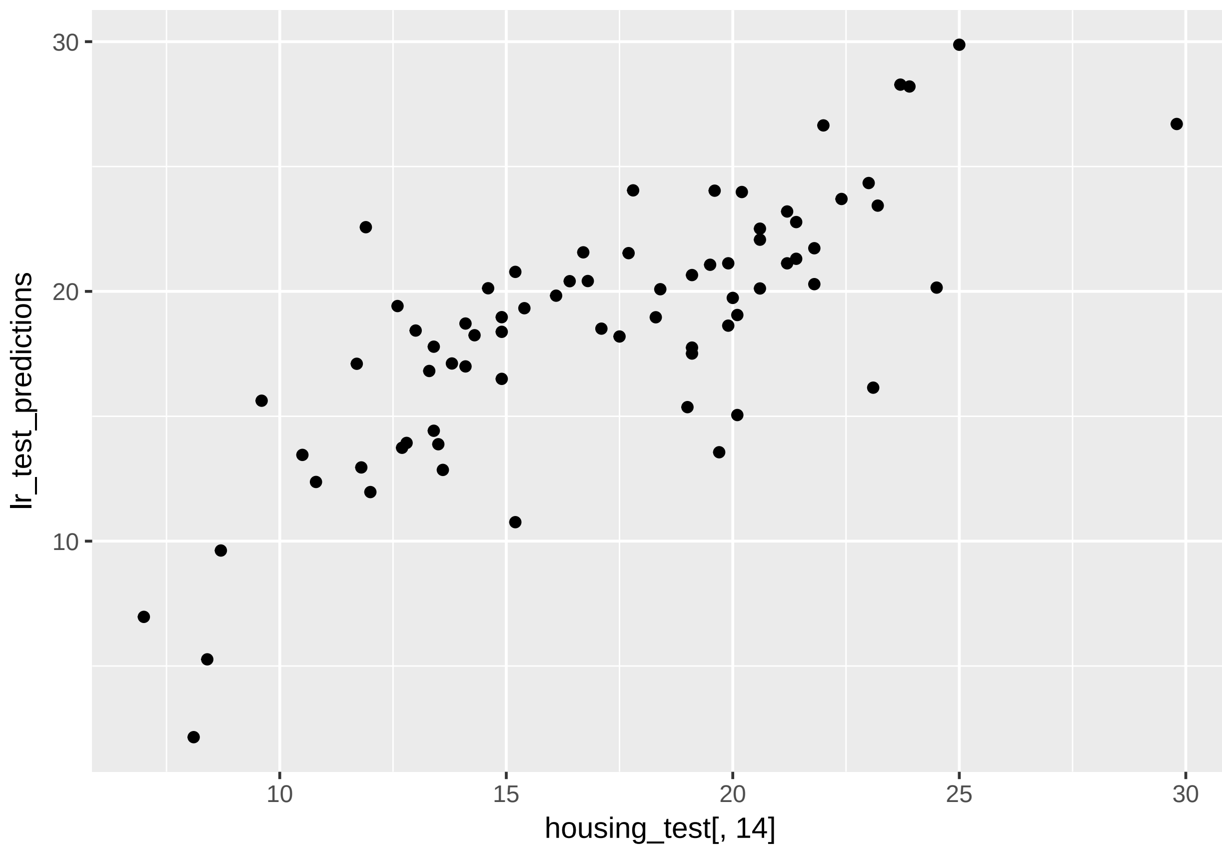
Weights: 32.0774107, -0.1220567, 0.0482206, 0.035769, 2.2280462, -15.454646, 4.0361421, 0.0112587, -1.3455648, 0.4004031, -0.0156138, -0.8786895, 0.0094675, -0.5573415 Training error: 23.7801251 Test error: 13.1967341

Oddly, the test error is better for this set, possibly because the test set does not include the any of the high value outliers that cause high error in the training set.

```
df1 <- data_frame(housing_train[,14],lr_train_predictions)

df2 <- data_frame(housing_test[,14],lr_test_predictions)

ggplot(df1,aes(x=housing_train[,14],y=lr_train_predictions))+
  geom_point()+
  geom_smooth(method='lm')
```




```
ggplot(df2,aes(x=housing_test[,14],y=lr_test_predictions))+
  geom_point()
```



1.3 Online gradient descent

a) Implement procedure

A function for normalizing a vector to values between 0 and 1.

```
normalize <- function(data){
  min <- min(data)
  range <- max(data)-min
  n <- length(data)

  normaldata <- integer(n)

  for(i in 1:n){
    point <- data[i]
    if(point==0){
      normaldata[i] <- 0
    }else{
      normaldata[i] <- (point-min)/range
    }
  }

  return(normaldata)
}
```

Then the actual gradient descent function.

```r
gd_online <- function(Xtrain,Xtest,ytrain,ytest,stepnum){
  d <- length(Xtrain[1,])
  n <- length(ytrain)

  XwithInt <- as.matrix(add_column(Xtrain, intercept=1, .before=1))

#initialize weights
  gd_weights <-integer(d+1)

#initialize empty df
  errors <- data_frame(iter=1:stepnum,trainerror=0,testerror=0)

  for (t in 1:stepnum){
#select a datapoint
        i <- t%%n
        if(i==0){i <- n}
#set a learning rate
        a <-  .1 #2/t

#Update Weight vector
    #helpful precalculation of error for this point y-f(x_i,w)
    point_error <-ytrain[i]-LR_predict(Xtrain[i,],gd_weights)

#    gd_weights <- gd_weights+(a*point_error)*XwithInt[i]

#update intercept
    gd_weights[1] <- gd_weights[1]+(a*point_error)

#update other weights
    for (j in 2:d+1){
        gd_weights[j] <- gd_weights[j]+(a*point_error*Xtrain[i,j-1])
    }

#    print(gd_weights)
    train_predictions <- LR_predict(Xtrain[,1:13],gd_weights)
    test_predictions <- LR_predict(Xtest[,1:13],gd_weights)

    trainerror <- MSE_calc(train_predictions,ytrain)
    testerror <- MSE_calc(test_predictions,ytest)

    errors[t,'trainerror'] <- trainerror
    errors[t,'testerror'] <- testerror
  }
  returnvals <- list(errors, gd_weights)
return(returnvals)
}
```

b) main3_3

The gradient descent function as written above is applied to normalized data below.

```r
  housing_train <- housing[1:433,]
  housing_test <- housing[434:506,]

  Xtrain <- housing_train[,1:13]
  Xtest <- housing_test[,1:13]
  ytrain <- housing_train[,14]
  ytest <- housing_test[,14]
  d <- ncol(Xtrain)
  ntrain <- nrow(Xtrain)
  ntest <- nrow(Xtest)

#normalize data
normalXtrain <-  data.frame(matrix(ncol = d, nrow = ntrain))
  for(col in 1:d){
    normalXtrain[,col] <- normalize(Xtrain[,col])
  }

normalXtest <-  data.frame(matrix(ncol = d, nrow = ntest))
  for(col in 1:d){
    normalXtest[,col] <- normalize(Xtest[,col])
  }

gd_output <- gd_online(normalXtrain,normalXtest,ytrain,ytest,1000)
```

The resulting errors were:

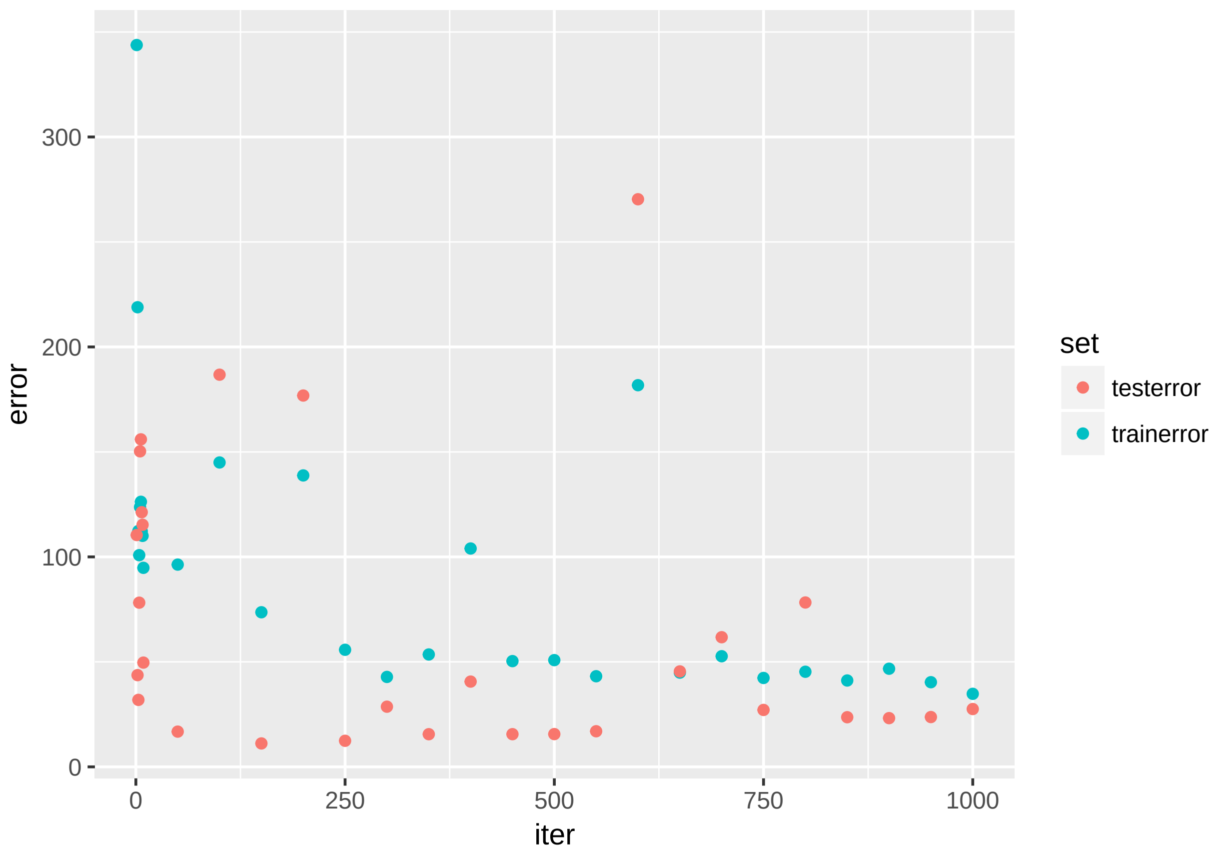
1. a=2/t

- training error: 59.2
- test error 69.3

1. a=.1

- training error: 34.8
- test error 27.5

```r
gd_errors <- gd_output[[1]] %>%
  gather(set, error, trainerror:testerror) %>%
  filter(iter<10|iter%%50==0)
gd_weight <- gd_output[[2]]

errorplot <- ggplot(gd_errors, aes(x=iter, y=error, color=set))+
  geom_point()

finalerrors <- gd_errors %>%
  filter(iter==1000)

errorplot
```



c) Un-normalized data

This accidentally made it into my code above, and ruined my life for a week. Using weights trained on normal data on unnormal data results in complete disaster.

```
#UNNORMALIZED DATA
gd_test_predictions <- LR_predict(housing_test[,1:13],gd_weight)
gd_test_predictions
```

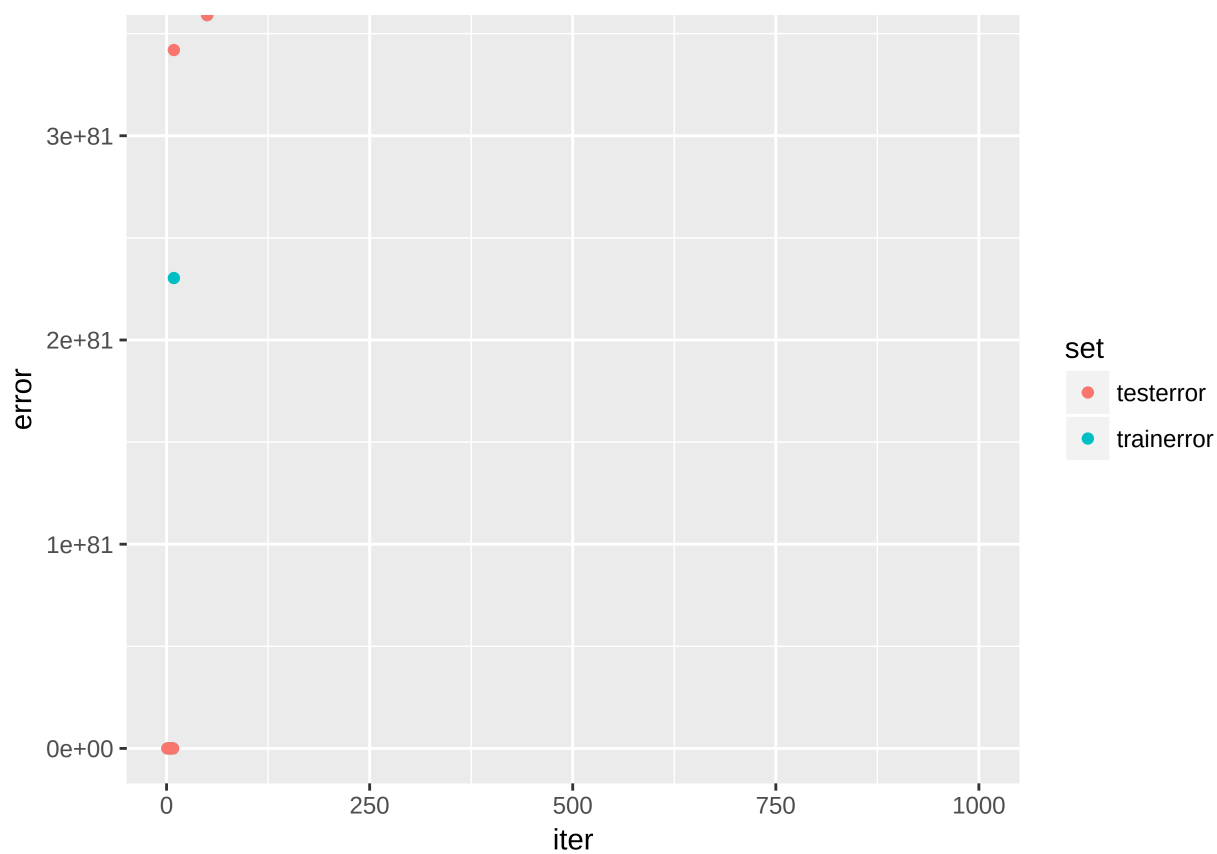
```
##  [1] -1879.399418 -1873.992351 -1909.532878 -2556.983138 -2864.578789
##  [6] -2442.789395   613.010001   582.931562   579.742129   703.597071
## [11]   594.688916  -781.929156 -2514.209862     6.321458   647.925821
## [16]   691.011012  -140.719198 -2781.026721   339.973744   612.283026
## [21]   550.699210 -2747.519778 -2340.743626 -2725.255401 -2744.674460
## [26]  -357.352961   762.523451  -510.950720   715.470060   780.478119
## [31]   804.962264   815.354376   264.592825 -2590.762382    63.301919
## [36]   475.901306   793.735669   739.306060   770.176435   757.064760
## [41]   656.626935   287.984576  -232.579412   689.600577   150.157480
## [46]   543.433411   666.047561   854.834256   863.630668   897.800635
## [51]   849.243623   608.211817   803.533449   726.772499   777.314436
## [56]   421.041357  -130.935061  -452.656919   378.880105   534.401606
## [61]  1863.851807  1860.242131  1779.923242  1686.795602  1808.222381
## [66]  1839.642935  1774.472148  1796.779981  2288.217008  2320.290381
## [71]  2371.049023  2325.868282  2330.915739
```

```
gd_train_predictions <- LR_predict(housing_train[,1:13],gd_weight)
```

Training and and evaluating entirely with unnormalized data also causes problems, quickly reaching unreasonable values which ggplot stumbles over...

```
  housing_train <- housing[1:433,]
  housing_test <- housing[434:506,]

  Xtrain <- housing_train[,1:13]
  Xtest <- housing_test[,1:13]
  ytrain <- housing_train[,14]
  ytest <- housing_test[,14]
  d <- ncol(Xtrain)
  ntrain <- nrow(Xtrain)
  ntest <- nrow(Xtest)

gd_output <- gd_online(Xtrain,Xtest,ytrain,ytest,1000)

gd_errors <- gd_output[[1]] %>%
  gather(set, error, trainerror:testerror) %>%
  filter(iter<10|iter%%50==0)
gd_weight <- gd_output[[2]]

errorplot <- ggplot(gd_errors, aes(x=iter, y=error, color=set))+
  geom_point()

finalerrors <- gd_errors %>%
  filter(iter==1000)

errorplot
```

```
## Warning: Removed 38 rows containing missing values (geom_point).
```



d)

Training and test errors for a constant learning rate are included above. the constant rate seems to have been much more effective in the end, though possibly less consistent in later iterations (this is hard to see becuase of the difference in scale… between plots. I'll include these plots if I have time to reorganize my code and save results under different variables later)

1.4

a) extendx

```
extendx <- function(X){
  newX <- X%>%
    mutate_all(funs(squared=.^2))
  return(newX)
}

extendedTrain <- extendx(housing_train[,1:13])
extendedTest <- extendx(housing_test[,1:13])
yTrain <- housing_train[,14]
yTest <- housing_test[,14]
```

b) Binary

The binary attribute stayed the same. 1^2=1 and 0^2=0

c) Extended regression

Again, ideally one would solve for the weights according to the formula copied above, but that math is not working for me, and I've used R's linear regression to cope. There's some added shenanigans below to deal with the fact that R removes chas_squared from the weight list, because it is redundant to chas (becuase it is binary), but generally, this is the same model as before.

Training error for this model is Xtrainerror 15, test error is 45.

```
LR_solve26 <- function(X,y){
  model<- lm(y~X[,1]+
       X[,2]+
       X[,3]+
       X[,4]+
       X[,5]+
       X[,6]+
       X[,7]+
       X[,8]+
       X[,9]+
       X[,10]+
       X[,11]+
       X[,12]+
       X[,13]+
       X[,14]+
       X[,15]+
       X[,16]+
       X[, 17]+
       X[,18]+
       X[,19]+
       X[,20]+
       X[,21]+
       X[,22]+
       X[,23]+
       X[,24]+
       X[,25]+
       X[,26]
     )
  w <- summary(model)$coefficients[,'Estimate']
  return(w)
}

weights <- LR_solve26(extendedTrain,yTrain) %>%
  as_data_frame() %>%
  add_row(value=0,.after = 16) %>%
  as_vector()

train_predictions <- LR_predict(extendedTrain,weights)
test_predictions <- LR_predict(extendedTest,weights)

Xtrainerror <- MSE_calc(train_predictions,yTrain)
```

```r
Xtesterror <- MSE_calc(test_predictions,yTest)

Xtrainerror
```

```
## [1] 15.23163
```

```r
Xtesterror
```

```
## [1] 45.0361
```

d) Report

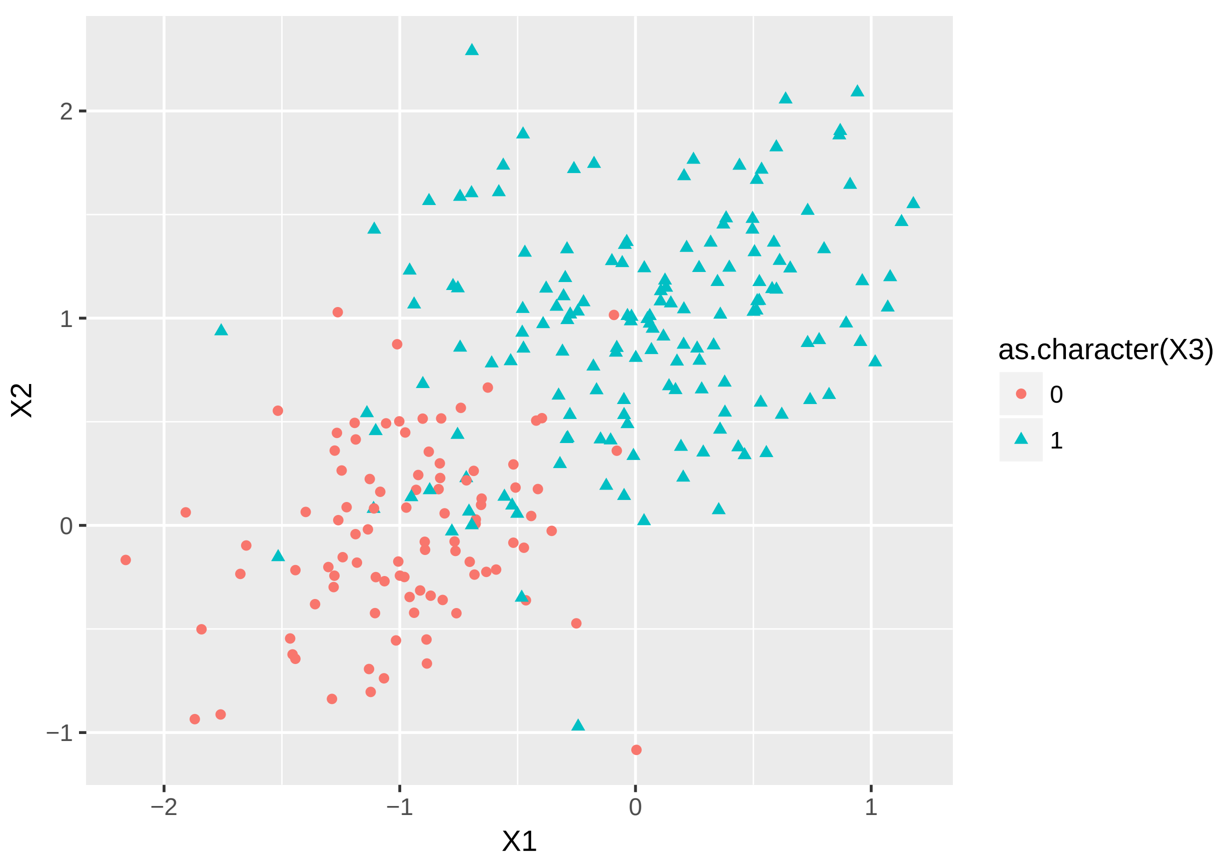
Training error: 15.2316305 Test error: 45.036098

Training error went down and test error increased, suggesting that this model may have overfit. The model that is linear in x will likely have better generalization error and is thus the better model according to this data.

Problem 2

2.1 Data Analysis

a) Plot



b) Report

These categories are not perfectly separable with a linear decision boundary.

2.2 Logistic regression

a) Derive Gradient of log likelihood

Given the log liklihood of a data set given parameters as below:

$$-log(p(D, \mathbf{w}) = \sum_{i=1}^{n} y_i log(\mu_i) + (1 - y_i) log(1 - \mu_i)$$

Derive the gradient of said log likelihood:

$$-\frac{\delta}{\delta w_j} - log(p(D, \mathbf{w}) = \sum_{i=1}^{n} x_{i,j}(y_i - g(z_i))$$

b) GLR

Something is wrong in this function, but I'm tired and giving up. Happy weekend.

```r
logr_predict<- function(X,logr_weights){
  X <- as.matrix(X)


  num_obs <-1
  num_obs <-nrow(X)
  num_feature <- ncol(X)

  z <- integer()
  prob <- integer()
  class <- integer()

  for(i in 1:num_obs){
    z[i] <- logr_weights %*% X[i,]
    print(z[i])
    prob[i] <- 1/(1+exp(-z[i]))
    if(prob[i] > .5){
      class[i] <- 1
    }
    else{class[i] <- 0}
  }

  return(class)
}



GLR <- function(Xtrain,Xtest,ytrain,ytest,stepnum){
  d <- ncol(Xtrain)
```

```
  ntrain <- nrow(ytrain)

  XwithInt <- as.matrix(add_column(Xtrain, intercept=1, .before=1))

#initialize weights
  logr_weights <-rep(1,d+1)

  Xtrain <- as.matrix(Xtrain)

  y <- as.matrix(y)

#initialize empty df
  errors <- data_frame(iter=1:stepnum,trainerror=0,testerror=0)

  for (k in 1:stepnum){

#select a datapoint
        i <- k%%ntrain
        if(i==0){i <- ntrain}

#make prediction
        guessvect <- logr_predict(XwithInt[i,],logr_weights)
        guess <- guessvect[1]
        incorrect <- ytrain[i]-guess

        if (incorrect){
        #set a learning rate
                a <- 2/k
        #Update Weight vector
            for (j in 1:d+1){
                logr_weights[j] <- logr_weights[j]+(a*XwithInt[i,j])
            }

        #                logr_weights <- logr_weights+(a*XwithInt[i])
        }
    }
return(logr_weights)
}

#    train_predictions <- LR_predict(Xtrain[,1:13],gd_weights)
#    test_predictions <- LR_predict(Xtest[,1:13],gd_weights)

#    trainerror <- MSE_calc(train_predictions,ytrain)
#    testerror <- MSE_calc(test_predictions,ytest)
#
#    errors[t,'trainerror'] <- trainerror
#    errors[t,'testerror'] <- testerror
#  }
#  returnvals <- list(errors, gd_weights)
#return(returnvals)
```

c) main2

```
trainX<- class_train[,1:2]
trainy <- class_train[,3]
testX <- class_test[,1:2]
testy <- class_test[,3]

#GLR(trainX,testX,trainy,testy,500)
```

2.3 Generative model

I lost an hour of work to a keyboard shortcut. Please forgive my brevity.

a) Class Conditional ML

For a classes 1 ($t_n = 1$)and 2 ($t_n = 0$), the class conditional ML estimates of μ is given (per Bishop 4.75 and 4.76) by

$$\mu_1 = \frac{1}{N_1}\sum_{n=1}^{N}(t_n)\mathbf{x}_n$$

$$\mu_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)\mathbf{x}_n$$

b) Covariance Matrix

The calculation of Σ is given in Bishop between 4.71 and 4.80. The process is to take the derivative of the log likelihood estimate with respect to Σ. The result is given below:

$$\mathbf{\Sigma} = \mathbf{S}$$

$$S = \frac{N_1}{N}\mathbf{S_1} + \frac{N_2}{N}\mathbf{S_2}$$

$$\mathbf{S_1} = \frac{1}{N_1}\sum_{n\in C_1}(\mathbf{x}_n - \mu_\mathbf{1})(\mathbf{x}_n - \mu_\mathbf{1})^T$$

$$\mathbf{S_2} = \frac{1}{N_2}\sum_{n\in C_2}(\mathbf{x}_n - \mu_\mathbf{2})(\mathbf{x}_n - \mu_\mathbf{2})^T$$

S1 | S |
S | S2 |

c) Prior

The class prior is also a Bernoulli distribtuion, the MLE of which is by now familiar.

$$\theta_{c=1} = \frac{N_1}{N}$$

d) Max_likelihood function

```
X <- class_train
y <- trainy

Max_Likelihood <- function(X){

  class1 <- X %>%
    filter(X3==1)

  class2 <- X %>%
    filter(X3==0)

  N <- X %>%
    count(X3)
  N1 <- N[2,'n']
  N2 <- N[1,'n']

  sumX1 <- class1 %>%
    summarise(sumX1=sum(X1),sumX2=sum(X2))

  sumX2 <- class2 %>%
    summarise(sumX1=sum(X1),sumX2=sum(X2))

  mu1 <-  as.matrix(1/N1)%*%as.matrix(sumX1)
  mu2 <-  as.matrix(1/N2)%*%as.matrix(sumX2)

  var01 <- class1 %>%
    mutate(X1=(X1-mu1[1])^2,X2=(X2-mu1[2])^2) %>%
    mutate(Z=X1+X2)
  var02 <- class2 %>%
    mutate(X1=(X1-mu2[1])^2,X2=(X2-mu2[2])^2)%>%
    mutate(Z=X1+X2)

  sig1 <- var01 %>%
    summarise(total=sum(Z)*1/N1)

  sig2 <- var02 %>%
    summarise(total=sum(Z)*1/N2)


  theta=N1/(N1+N2)

  bigTheta <- list(mu2, sig2,mu1,sig1,theta)
  return(bigTheta)
}
```