Homework 4

Amit Arora

July 26, 2018

Linear Model: converting Celcius to Fahrenheit

Since we talked about this a lot during class so lets try to do this as an example. The solution is presented in the appendix but try to do this as much as possible on your own before looking at the solution.

1. Create 100 random values between 0 to 100 using the *runif* function (see help on this function for more information on usage), these numbers represent (lets say) temperature in celcius. Store these numbers in a variable called c.
2. Now create another variable called f for temperature in Fahrenheit which contains the values of the temperature in c but converted to Fahrenheit i.e. $f = 1.8c + 32$.
3. Now we know the true relationship between f and c here but lets have ML model try to determine it using the data we just generated. So use the **lm** functiom (usage is similar and simpler than glm that we used in class) to create a linear model which will give you the mapping between f and C. Print out the coefficiencts of the model and now can you write the equation of the model? Is it similar to the equation of the true model?
4. [Bonus question] Lookup help online to see if you can plot a scatter plot to show the original data and then the straight line showing the ML model that you determined in step 3.

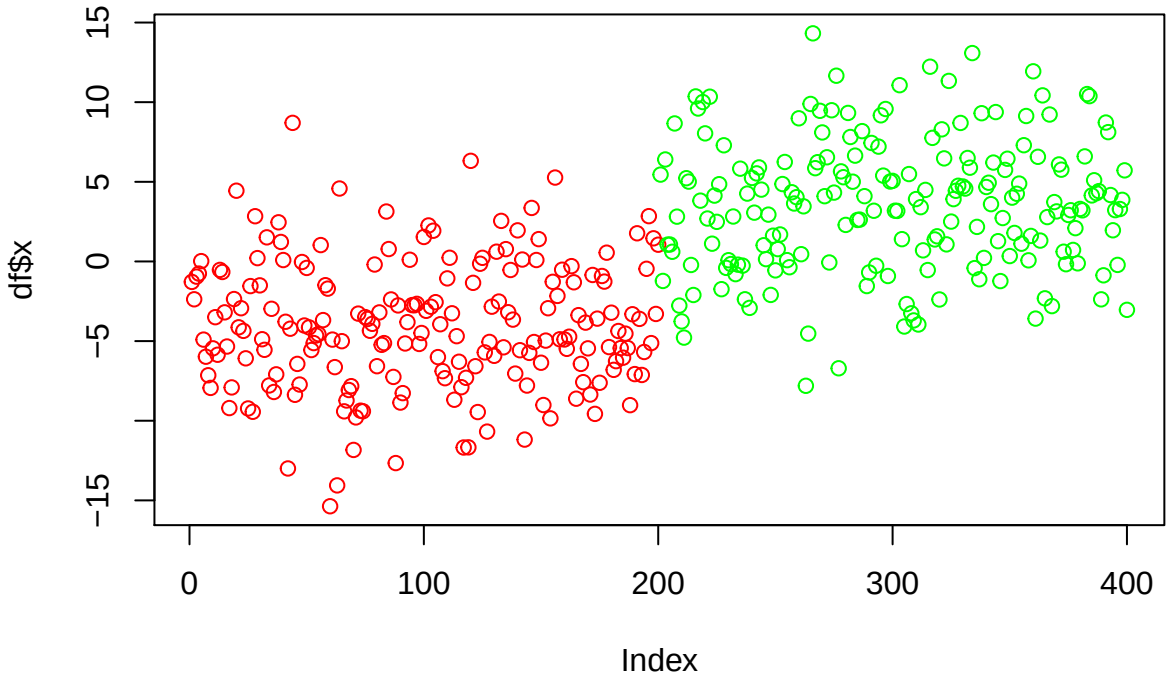
Another classification problem

This is a simple example of the classification that we are discussing in class. Suppose we have a dataset which has a mixture of two types of measurement (does not matter what they represent) and we want to separate them out.

1. We take 200 random samples from a Normal distribution (see function *rnorm*) with mean 4 and standard deviation 4. Now take another 200 samples from another Normal distribution with mean -4 and standard deviation 4. Plot this data as a scatter plot and color code them as red and green. The code is provided below.
2. Use the glm function to determine a model which will be able to separate out the distributions.
3. [Bonus Question] Do a 80/20 train test split and now determine the training and test accuracy, confusion matrix.

Here is some code to get you started.

```
B = 200
x1 = rnorm(B, -4, 4)
x2 = rnorm(B, 4, 4)
df = data.frame(x = c(x1, x2), y = as.factor(c(rep(1, B), rep(0, B))))
plot(df$x, col=ifelse(df$y == 1, 'red', 'green'))
```




```
# write the code for the glm, print model summary, do predictions on training data
```

Problem 3 Neural networks

Neural networks are *universal approximators* i.e. they can be used to approximate any continous function over a compact set (i.e. closed and bounded). See this link and this link.

Here we consider a very simple example illustrating that a neural network with no hidden layer is equivalent to a linear model (i.e. a linear equation) and then we extend the neural network to have one hidden layer with 3 units and see if it performs better than the linear model.

** Solution is provided in Appendix B. Please try to solve as many parts as you can on your own before looking at the solution **

1. Install the "nnet" and "NeuralNetTools" package.
2. Read the data from the file "data.csv". It contains 3 independant variables x, y and z and a target variable t.
3. Do a train/test split.
4. Make a linear model using the lm function, print the model summary using the summary function and then write the model as a linear equation (by extracting the coefficiencts from the model and using them to write "t" as a linear function of x, y and z).
5. Write an expression for the sum of squared errors for the training data set and the test data set.
6. Make a neural network using the nnet package, see help on nnet for details.
7. Write an expression for the sum of squared errors for the training data set and the test data set.

8. Draw the neural network using the plotnet function. See help on plotnet for details.
9. Are the linear equations from the neural net same as the equation from the linear model?
10. Change the number of neurons in the hidden layer to 3 i.e. add one hidden layer with 3 units between input and output. Rerun the neural network model.
11. Are the linear equations the same as in previous steps? Is the training SSE and test SSE better than what it was in the linear model and the no hidden layer neural network?

Appendix A: Code for Problem 1

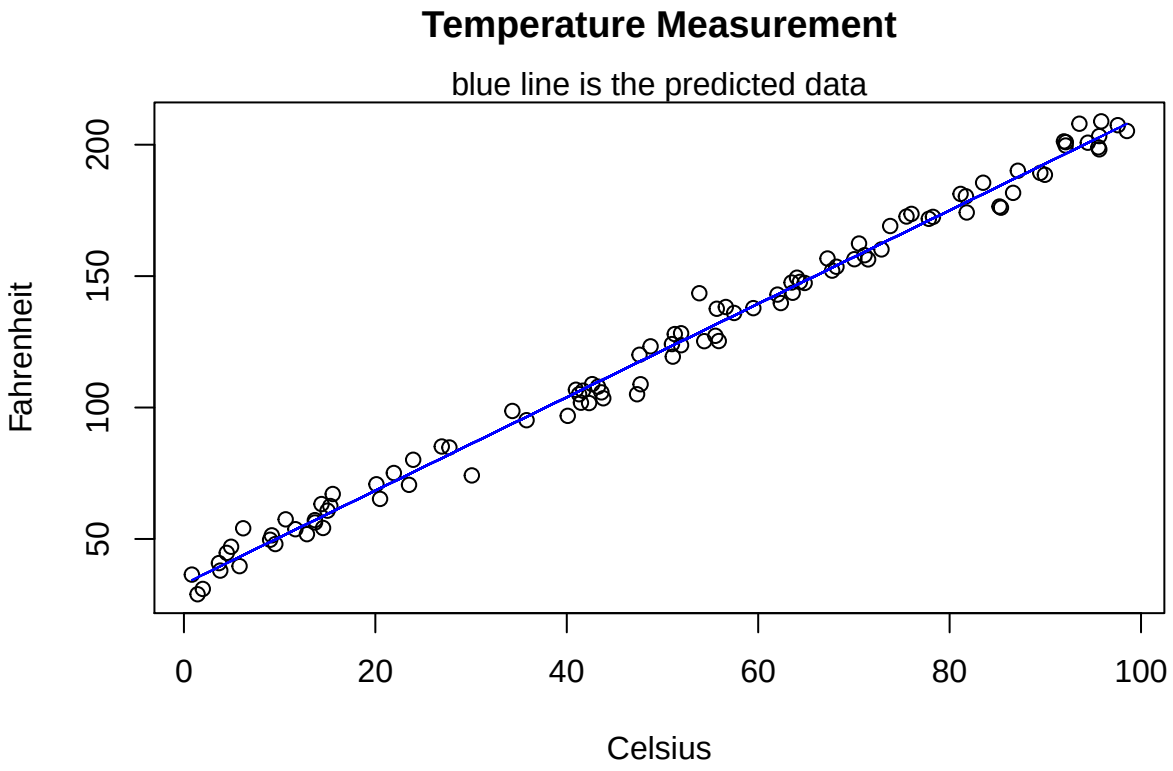
```
## simple linear model example
B = 100
c = runif(B, 0, 100)
f = 1.8*c + 32 + rnorm(B, 0, 5)
plot(x=c, y=f,
     xlab = "Celsius",
     ylab = "Fahrenheit",
     main = "Temperature Measurement")

fit = lm(f ~ c)
summary(fit)
```

```
##
## Call:
## lm(formula = f ~ c)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.1306  -3.6914   0.2041   3.4046  14.9401
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 32.84949    0.93018   35.31   <2e-16 ***
## c            1.77706    0.01588  111.90   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.68 on 98 degrees of freedom
## Multiple R-squared:  0.9922, Adjusted R-squared:  0.9922
## F-statistic: 1.252e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
slope = coefficients(fit)[2]
intercept = coefficients(fit)[1]

lines(x=c, y=slope*c + intercept, col='blue')
mtext("blue line is the predicted data")
```



Appendix B, Code for problem 3

NOTE: there are some warnings generated, disabling them for now.

```
# load neural network packages
#install.packages(c("nnet", "NeuralNetTools"))
library(nnet)
library(NeuralNetTools)
library(dplyr)

# set a random seed to get repeatable results
set.seed(1234)

linear_model <- function(df_train, df_test) {
  # create a linear model, we want to model the target variable "t"
  # using 3 predictors x, y and z
  lm_fit <-lm(t ~ x+y+z, data=df_train)
  summary(lm_fit)

  # we have the model now, it is a linear model so we write it as an
  # a linear equation in 3 variables x, y and z (which were used to create this model)
  cat(sprintf('Linear model is target = %0.4f + (%0.4fx) + (%0.4fy) + (%0.4fz))\n',
              lm_fit$coefficients[1], lm_fit$coefficients[2],
              lm_fit$coefficients[3], lm_fit$coefficients[4]))
```

```
  # lets  looking at the trainin SSE (this is what we minimized to get the parameters of the model)
  train_SSE = sum(lm_fit$residuals^2)
  cat(sprintf('Training Sum of Squared Errors (SSE)
              for linear model is %0.4f\n', train_SSE))

  # now lets predict on the "unseen" a.k.a test data
  predictions = predict(lm_fit, df_test)

  # how well did the model perform, lets see test SSE vs training SSE
  test_SSE = sum((predictions - df_test$t)^2)
  rmse = sqrt(mean((predictions - df_test$t)^2))
  cat(sprintf('Test Sum of Squared Errors (SSE) for linear model is %0.4f,
              training SSE was %.4f\n', test_SSE, train_SSE))
  cat(sprintf('Root mean squared error %.4f', rmse))
}

nn_model <- function(df_train, df_test, size=0) {
  # now lets try a neural network with no hidden layer (size = 0)
  # this is essentialy same as the linear model as we shall see
  # when look at the params of the model and the diagramatic representation of the
  # neural network
  nn_fit <- nnet(t ~ x+y+z,
                     size = size, data=df_train, skip = T,
                     trace = F,maxit = 1000, linout = T)
  summary(nn_fit)

  cat(sprintf('NN model is target = %0.4f + (%0.4fx) + (%0.4fy) + (%0.4fz))\n',
              nn_fit$wts[1], nn_fit$wts[2], nn_fit$wts[3], nn_fit$wts[4]))
  train_SSE = sum(nn_fit$residuals^2)

  cat(sprintf('Training SSE for nnet model is %0.4f\n', train_SSE))

  predictions = predict(nn_fit, df_test, type="r")
  test_SSE = sum((predictions - df_test$t)^2)
  rmse = sqrt(mean((predictions - df_test$t)^2))
  cat(sprintf('Test Sum of Squared Errors (SSE) for neural net model is
              %0.4f\n, training SSE was %.4f\n', test_SSE, train_SSE))
  cat(sprintf('Root mean squared error %.4f', rmse))

  plotnet(nn_fit)
}

# read the data
df = read.csv("data.csv")
sample_n(df, 10)
```

```
##         x    y    z    t
## 23   13.2 15.9 49.6  5.6
## 124 123.1 34.6 12.4 15.2
## 121 141.3 26.8 46.2 15.5
## 123 224.0  2.4 15.6 11.6
## 169 215.4 23.6 57.6 17.1
## 125 229.5 32.3 74.2 19.7
## 2    44.5 39.3 45.1 10.4
```

```
## 45   25.1 25.7 43.3  8.5
## 128  80.2  0.0  9.2  8.8
## 99  289.7 42.3 51.2 25.4
```

```
glimpse(df)
```

```
## Observations: 200
## Variables: 4
## $ x <dbl> 230.1, 44.5, 17.2, 151.5, 180.8, 8.7, 57.5, 120.2, 8.6, 199....
## $ y <dbl> 37.8, 39.3, 45.9, 41.3, 10.8, 48.9, 32.8, 19.6, 2.1, 2.6, 5....
## $ z <dbl> 69.2, 45.1, 69.3, 58.5, 58.4, 75.0, 23.5, 11.6, 1.0, 21.2, 2...
## $ t <dbl> 22.1, 10.4, 9.3, 18.5, 12.9, 7.2, 11.8, 13.2, 4.8, 10.6, 8.6...
```

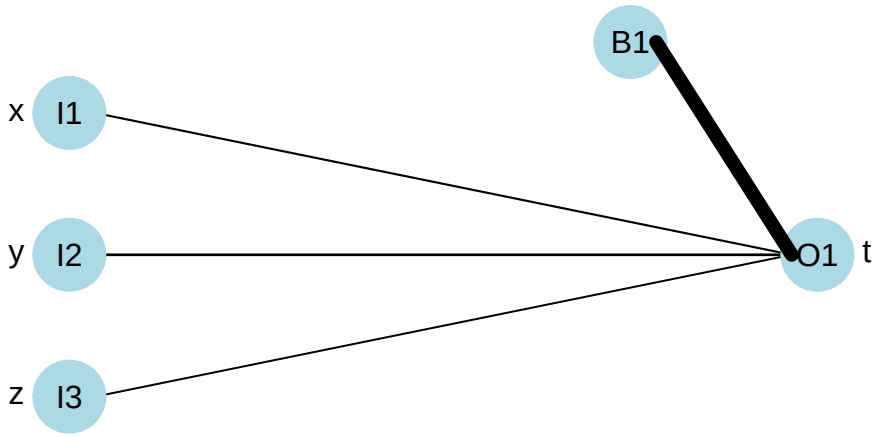
```
# create a test train split, 80/20 split being done here
n = nrow(df)
train = sample(n, 0.8*n)
df_train = df[train,]
df_test  = df[-train,]

linear_model(df_train, df_test)
```

```
## Linear model is target = 3.1184 + (0.0443x) + (0.1856y) + (0.0028z))
## Training Sum of Squared Errors (SSE)
##               for linear model is 441.5425
## Test Sum of Squared Errors (SSE) for linear model is 119.4414,
##               training SSE was 441.5425
## Root mean squared error 1.7280
```

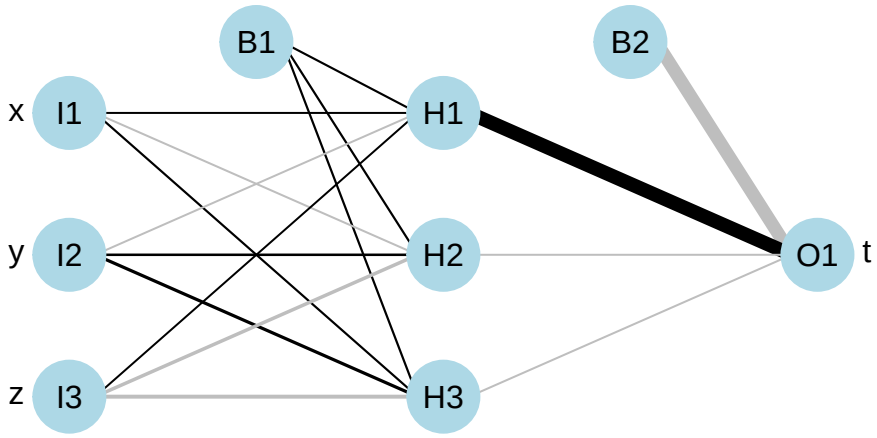
```
nn_model(df_train, df_test, 0)
```

```
## NN model is target = 3.1184 + (0.0443x) + (0.1856y) + (0.0028z))
## Training SSE for nnet model is 441.5425
## Test Sum of Squared Errors (SSE) for neural net model is
##               119.4414
## , training SSE was 441.5425
## Root mean squared error 1.7280
```




```
nn_model(df_train, df_test, 3)
```

```
## NN model is target = 2.9002 + (0.0065x) + (-0.0216y) + (0.0015z))
## Training SSE for nnet model is 77.4856
## Test Sum of Squared Errors (SSE) for neural net model is
##               21.6857
## , training SSE was 77.4856
## Root mean squared error 0.7363
```


B1
B2
x
I1
H1
y
I2
H2
O1
t
z
I3
H3