Lab 1

Linear regression

It’s a prediction model that use inputs to predict and produce output. The outputs are purely statistical numbers. We call the numbers we are given inputs and the numbers we predict outputs.

1. The baseline model

Baseline is a naive model in linear regression. It uses the mean value form the input to predict the output. For this chapter, we will use a dataset that involves longevity.

```
install.packages("ggplot2")
library('ggplot2')
ages <- read.csv(“longevity.csv”)
View(ages)

ggplot(ages, aes(x = AgeAtDeath, fill = factor(Smokes))) +
  geom_density() +
  facet_grid(Smokes ~ .)
```

We get the following graph:

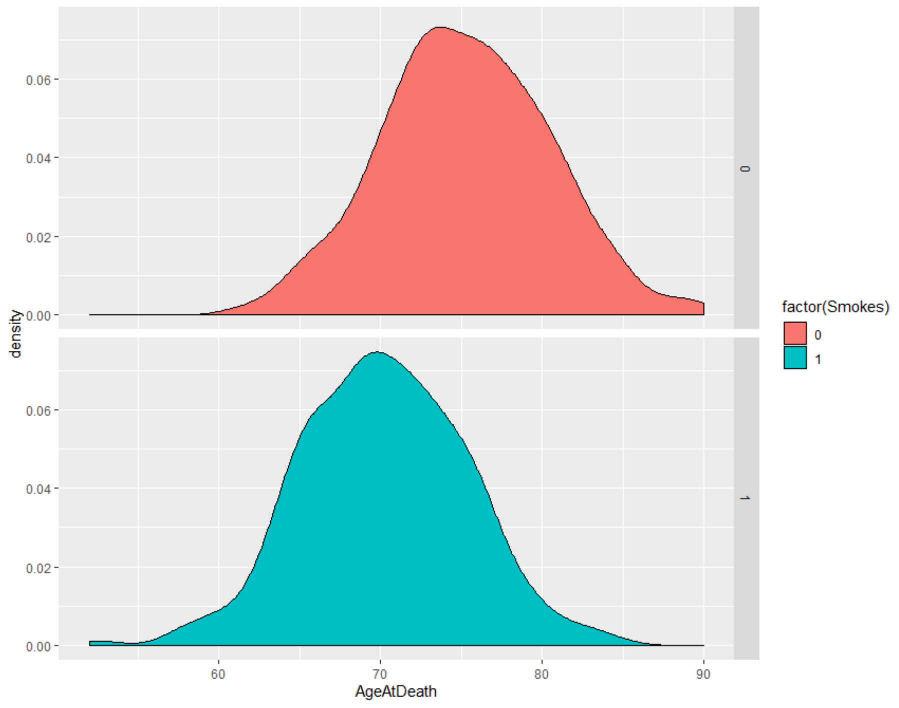

Figure: Density plot of 1,000 people’s life spans, facetted by smokers

2. Mean Squared Error (MSE)

Probably one of the most important number in statistics is the **squared error**. In which, the y is the hypothesis true prediction, and h is our guess. The equation for squared error is: (y - h) ^ 2.

For the data set we have, the summary as follows:

```
summary(ages$AgeAtDeath)
```

To demonstrate, let's make our h = 73 and calculate as follows:

```
guess <- 73
with(ages, mean((AgeAtDeath - guess)^2))
```

In the script, we can observe that our calculated **mean squared error (MSE)** value of the squared error to each of the data entry, which is 32.991.

In order to convince ourselves that h=73 is the best option, we can use the following code to examine the possible combinations of h and y:

```
guess.accuracy <- data.frame()
```

```
for (guess in seq(63, 83, by = 1))
{
  prediction.error <- with(ages,
                           mean((AgeAtDeath - guess) ^ 2))
  guess.accuracy <- rbind(guess.accuracy,
                          data.frame(Guess = guess,
                                     Error = prediction.error))
}


ggplot(guess.accuracy, aes(x = Guess, y = Error)) +
  geom_point() +
  geom_line()
```

As we can observe from the graph below, the lowest point is at our prediction of 73.

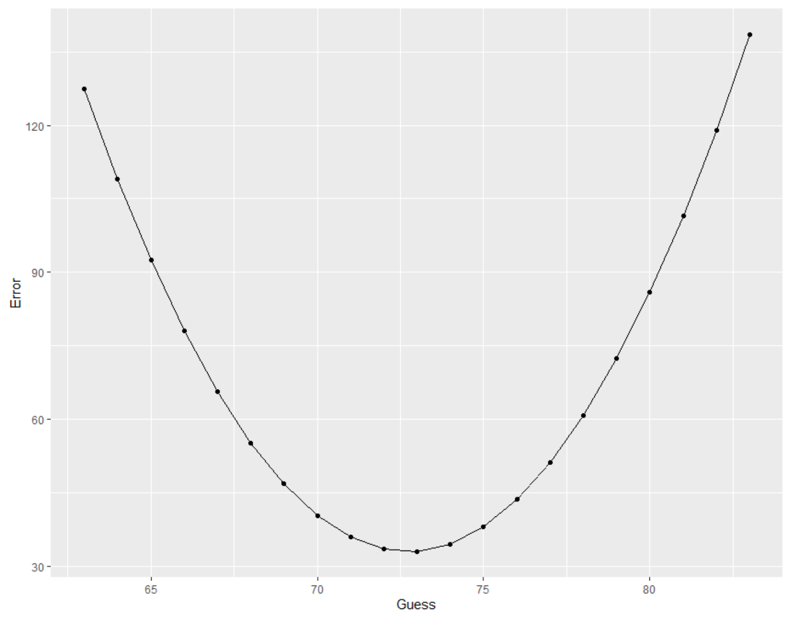

Figure: Mean squared error (MSE)

3. Root Mean Squared Error (RMSE)

Sometimes, MSE is quite large and hard to use. Therefore, we will use root mean squared error (RMSE) instead, and calculate the data set as following:

```
constant.guess <- with(ages, mean(AgeAtDeath))
```

```
with(ages, sqrt(mean((AgeAtDeath - constant.guess) ^ 2)))
```

```
smokers.guess <- with(subset(ages, Smokes == 1),
                      mean(AgeAtDeath))
```

```
non.smokers.guess <- with(subset(ages, Smokes == 0),
                          mean(AgeAtDeath))
```

```
ages <- transform(ages,
                  NewPrediction = ifelse(Smokes == 0,
                                         non.smokers.guess,
                                         smokers.guess))
```

```
with(ages, sqrt(mean((AgeAtDeath - NewPrediction) ^ 2)))
```

So the equation has changed to **sqrt(mean((AgeAtDeath – constant.guess)^2))**

The difference is: without smoke data, our prediction of RMSE is at 5.737096, if we factor in the smoke data, we would get 5.148622, the prediction value is better than just use **AgeAtDeath** alone. In other words, the latter is more accurate prediction. The point for RMSE is that it can incorporate multiple data to derive the RMSE. In our example, we used two: the **non-smoker.guess** and **smoker.guess** set like the following. Of cause, we can add more to it, if we can come up with additional variables. That way, the prediction may become more accurate. (Assuming we are using the relevant variables derived from correct calculations)

4. Linear regression in a nut shell

Let’s consider the following:

```
heights.weights <- read.csv("01_heights_weights_genders.csv")
```

```
ggplot(heights.weights, aes(x = Height, y = Weight)) +
  geom_point() +
  geom_smooth(method = 'lm')
```

The thing to notice here is the **`geom_smooth`** function which call upon ‘lm’, short for “linear models”.

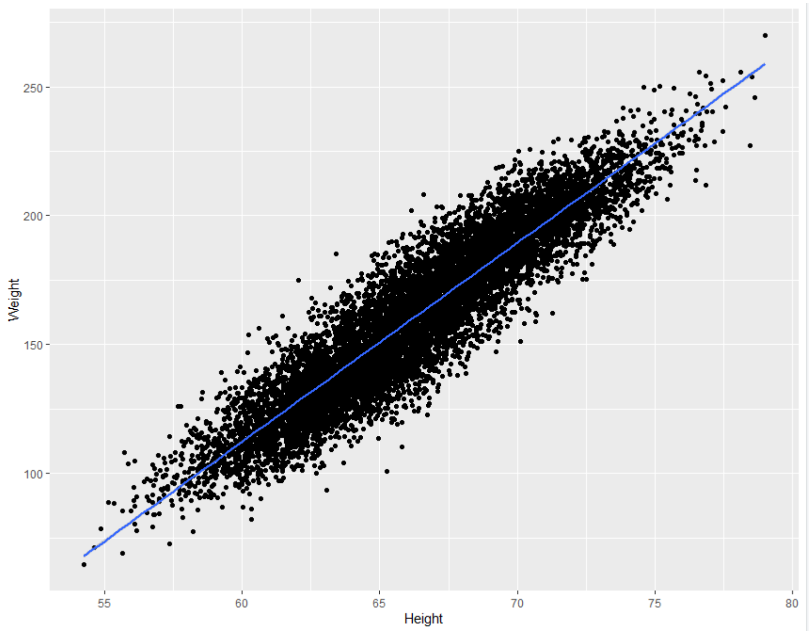

Figure: Weights versus heights with the regression line added

5. Coefficients

R is incredibly convenient in machine learning. We can call **lm** to get the number for that line directly. All we need to do is to specify a regression formula using the ~operator. For this example, we are predicting weight from height, so we use **Height~Weight**. So, we have the following:

```
fitted.regression <- lm(Weight ~ Height,
                        data = heights.weights)

coef(fitted.regression)
```

The answer is:

```
 (Intercept) Height

-350.737192 7.717288
```

Pull out the intercept of the regression line with a call to the **coef** function, this will return the coefficients for a linear model. (Keep in mind that we are talking about linear regression models here, there are other regression models.)

```
intercept <- coef(fitted.regression)[1]
slope <- coef(fitted.regression)[2]
```

Hence,

```
predicted.weight <- intercept + slope * observed.height,

predicted.weight == -350.737192 + 7.717288 * observed.height
```

This shade some lights in the coefficients of the **fitted.regression**. Basically, for every inch, we should observe an increase of 7.7 pounds. The intercept means that at height 0 inches, we predict the person weight -350 pounds.

If we trace along the line, we can see if a person weigh 0 pounds, the height should be 45 inches tall. Therefore, our model doesn't really work for children or extremely short adults.

To find out what's wrong with our prediction, we need to use **predict** function to find the model's prediction.

```
predict(fitted.regression)
```

6. Residuals

The predict function doesn’t really give us anything useful, so we need to dig a little bit deeper.

```
true.values <- with(heights.weights, Weight)
errors <- true.values - predict(fitted.regression)
```

We compare the true values from our inputs to our prediction of our regression model. And what output we got from errors is called **residuals**, it’s the same results by residuals function.

```
residuals(fitted.regression)
```

7. R-squared

What we really want is: for linear regression. This tells you how much better your models does than just using the mean function alone. The number range between 0 and 1, thus gives you an indicator how much better (far away from 0) you are doing than just using mean.

```
mean.mse <- 1.09209343
model.mse <- 0.954544

r2 <- 1-(model.mse /mean.mse)

r2
```

Judging by the result, we can say that our regression accounts for roughly 12.6% of the variance.