R Notebook

Chick Weight and Diet

This study determines which diet results in the largest weight gain in chicks. Diet1 is the normal diet and the other three diets are alternatives. We use non-parametric methods to determine: - Which of the diets resulted in statistically most significant weight gain compared to Diet1 - Provide a confidence interval for an estimate of the real weight gain - Determine if a quadratic regression model is significantly more accurate than a linear regression model.

2.2.1 Data preparation

5 of the chicks do not have measurements for the final day, day 21 so were removed from the analysis.

```
chickweight = read.csv("chick_weight.csv", row.names = 1)
head(chickweight)
```

```
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1
```

```
dim(chickweight)
```

```
## [1] 578   4
```

```
# Every chick is already assigned to a diet.
# We will simply identify which chicks are missing a value for day 21
# And then remove those chicks from the original dataframe.
# We pivot the long format to wide so that we have one row per chick.
library(reshape2)
chick_wide = dcast(chickweight, Chick ~ Time, value.var="weight")
dim(chick_wide)
```

```
## [1] 50 13
```

```
# Any row that is missing a value is eliminated.
chick_wide_nona = na.omit(chick_wide)
dim(chick_wide_nona)
```

```
## [1] 45 13
```

```r
# A quick way to look at the indexes of rows (each representing a chick) without missing values
chick_wide_nona$Chick
```

```
##  [1]  1  2  3  4  5  6  7  9 10 11 12 13 14 17 19 20 21 22 23 24 25 26 27 28 29
## [26] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 45 46 47 48 49 50
```

```r
chickweight = chickweight[chickweight$Chick %in% chick_wide_nona$Chick,]
dim(chickweight)
```

```
## [1] 540   4
```

Get Final Weights

```r
# For all the diets, obtain the final weights.
# We could have simply selected day 21, however this function allows
# to obtain final weights even if the chick does not have a measurement
# for day 21.
get_final_weights_for_diet <- function(diet){
    df_this_diet = subset(chickweight, Diet==diet)
    final_weights = numeric()
    chicks = unique(df_this_diet$Chick)
    for (chick_num in chicks) {
        temp_df = subset(df_this_diet,Chick==chick_num)
        final_weight = temp_df$weight[length(temp_df$weight)]
        final_weights = c(final_weights,final_weight)
    }
    return(final_weights)
}

all_final_weights = list()
# retrieve final weights for each chick. The result is a dictionary where diet is the key and the value
for(i in 1:4){
    all_final_weights[[i]] = get_final_weights_for_diet(i)
}

all_final_weights
```

```
## [[1]]
##  [1] 205 215 202 157 223 157 305  98 124 175 205  96 266 142 157 117
##
## [[2]]
##  [1] 331 167 175  74 265 251 192 233 309 150
##
## [[3]]
##  [1] 256 305 147 341 373 220 178 290 272 321
##
## [[4]]
## [1] 204 281 200 196 238 205 322 237 264
```

The Shuffle function

```r
# This shuffle function takes any number of groups. In this case each group correspondence to the value
# Given several groups, it will shuffle the values among the groups. In our analysis we perform shuffle

shuffle <- function(grps){
  num_grps = length(grps)

  # pool all values
  pool = unlist(grps)

  pool = sample(pool, length(pool), replace=F)

  # reassign to groups of same size as original groups
    new_grps = list()
    start_index = 1
    end_index = NA
    for(i in 1:num_grps){
        end_index = start_index + length(grps[[i]]) - 1
        new_grps[[i]]=pool[start_index:end_index]
        start_index = end_index+1

    }
    return(new_grps)

}
```

```r
# A quick function to return the difference of the average of two groups
meandiff <- function(grpA, grpB){
    return ((sum(grpB) / length(grpB)) - (sum(grpA) / length(grpA)))

}
```

```r
# this function draws a histogram.
# The diet_1 and diet_2 arguments are the names of the two diets being compared.
# The arguments diet_1 and diet_2 are used simply for the title of the plot
# the observed argument defines where to draw the vertical line
# the d argument is a set of values.
# This function was used to create Figure 2.1 - Histogram of distribution of differences
library(ggplot2)
draw_hist <- function(d, observed, diet_1, diet_2) {
  qplot(d, geom="histogram",
        xlab="Difference",
        ylab="Frequency",
        main=paste("Distribution of Difference Diet ",
                   diet_1, " - Diet ", diet_2 , sep=""),
        fill=I('#0504aa')
  )

}
```

```r
# This is the main function
# - first we calculate the observed mean of difference
# - Then 10,000 times, we shuffle the samples
# - calculate the difference of the means of the shuffled samples
```

```
#   and compare them to the difference of the means observed
#
# This is a one-tailed test, because we are looking only for an increase
#   or positive change when comparing weights of diet2-4 compared to
#   normal diet 1

diff2meansig <- function(diet_1, diet_2, grpA, grpB){
    # list of lists
    samples = list(grpA, grpB)
    a = 1
    b = 2

    observed_mean_diff = meandiff(samples[[a]], samples[[b]])
    all_mean_diffs = numeric()

    count = 0
    num_shuffles = 10000

    for(i in 1:num_shuffles) {
        new_samples = shuffle(samples)
        mean_diff = meandiff(new_samples[[a]], new_samples[[b]])
        all_mean_diffs[i]=mean_diff
        if (mean_diff >= observed_mean_diff) {
            count = count + 1
        }
    }
    draw_hist(all_mean_diffs, observed_mean_diff, diet_1, diet_2)


    ######################################
    #
    # Output
    #
    ######################################
    pvalue=count/num_shuffles

    print(paste("**********Diet: ", diet_1, " vs Diet: ", diet_2,"**********",
                sep=""))

    print(paste("Observed difference of two means: ", observed_mean_diff, sep=""))
    print(paste(count, " out of ", num_shuffles,
                " experiments had a difference of two means greater than ",
                " or equal to observed_mean_diff ", observed_mean_diff, ".", sep=""))
    print(paste("The chance of getting a difference of two means ",
                "greater than or equal to ", observed_mean_diff, " is ",
                pvalue, ".", sep=""))


  }
```

```
set.seed(123) # this step is not necessary.
              # it simply guarantees to output the same results as in the book.
```

```
combinations = cbind(1,2:4)

for (i in 1:nrow(combinations)) {
  diet_1 = combinations[i,1]
  diet_2 = combinations[i,2]
  diff2meansig(diet_1, diet_2,
               all_final_weights[[diet_1]], all_final_weights[[diet_2]])
}
```

```
## [1] "**********Diet: 1 vs Diet: 2**********"
## [1] "Observed difference of two means: 36.95"
## [1] "925 out of 10000 experiments had a difference of two means greater than  or equal to observed_me
## [1] "The chance of getting a difference of two means greater than or equal to 36.95 is 0.0925."
## [1] "**********Diet: 1 vs Diet: 3**********"
## [1] "Observed difference of two means: 92.55"
## [1] "8 out of 10000 experiments had a difference of two means greater than  or equal to observed_mean
## [1] "The chance of getting a difference of two means greater than or equal to 92.55 is 8e-04."
## [1] "**********Diet: 1 vs Diet: 4**********"
## [1] "Observed difference of two means: 60.8055555555555"
## [1] "63 out of 10000 experiments had a difference of two means greater than  or equal to observed_mea
## [1] "The chance of getting a difference of two means greater than or equal to 60.8055555555555 is 0.
```

2.3.2 WEIGHT CHANGE CONFIDENCE INTERVAL

```
# The bootstrap function randomly selects
# values with replacement within a group
bootstrap <- function(x){
  samp_x = numeric()
  samp_x = sample(x, length(x), replace=T)

  return(samp_x)
}
```

```
# This is the main function for confidence interval.
# We bootstrap the values from within the individual groups
# and calculate the difference between the means.
# Then we sort the differences and take the 5th and 95th highest
# value for a 90% confidence interval

diff2meanconf <- function(diet_1, diet_2, grpA, grpB, conf_interval){
    # list of lists
    samples = list(grpA, grpB)
    a = 1
    b = 2

    observed_mean_diff = meandiff(samples[[a]], samples[[b]])

    num_resamples = 10000   # number of times we will resample from our original samples
    out = numeric()                 # will store results of each time we resample

    for (i in 1:num_resamples){
```

```
        # get bootstrap samples for each of our groups
        # then compute our statistic of interest
        # append statistic to out
        bootstrap_samples = list()  # list of lists
        for (j in 1:length(samples)){
            bootstrap_samples[[j]]=bootstrap(samples[[j]])
        }

        # now we have a list of bootstrap samples, run meandiff
        out[i]=meandiff(bootstrap_samples[[a]], bootstrap_samples[[b]])
    }

    out = sort(out, decreasing = F)

    tails = (1 - conf_interval) / 2

    # in case our lower and upper bounds are not integers,
    # we decrease the range (the values we include in our interval),
    # so that we can keep the same level of confidence
    lower_bound = ceiling(num_resamples * tails)
    upper_bound = floor(num_resamples * (1 - tails))

    ########################################
    #
    # Output
    #
    ########################################

    # print observed value and then confidence interval
    print(paste("**********Diet: ", diet_2, " - ", diet_1,  " **********", sep=""))
    print(paste("Observed difference between the means: ", observed_mean_diff, sep=""))
    print(paste("We have ", conf_interval * 100, "% confidence that the true difference between the mea
    print(paste("is between: ", round(out[lower_bound],2), " and ", round(out[upper_bound], 2), sep=""))
}
```

```
set.seed(123)

for (i in 1:nrow(combinations)) {
  diet_1 = combinations[i,1]
  diet_2 = combinations[i,2]
  diff2meanconf(diet_1, diet_2, all_final_weights[[diet_1]], all_final_weights[[diet_2]], conf_interval
}
```

```
## [1] "**********Diet: 2 - 1 **********"
## [1] "Observed difference between the means: 36.95"
## [1] "We have 90% confidence that the true difference between the means"
## [1] "is between: -8.8 and 80.86"
## [1] "**********Diet: 3 - 1 **********"
## [1] "Observed difference between the means: 92.55"
## [1] "We have 90% confidence that the true difference between the means"
## [1] "is between: 50.53 and 133.7"
## [1] "**********Diet: 4 - 1 **********"
## [1] "Observed difference between the means: 60.8055555555555"
```

```
## [1] "We have 90% confidence that the true difference between the means"
## [1] "is between: 28.58 and 93.19"
```

2.4 Regression Analysis

```
# given x and y coordinates and the degree of line to fit,
# the function will return the coefficients of the line
# For example, in a linear regression it will return slope and intercept.
# In a quadratic regression, it will be coefficients for quadratic term, linear term, and the intercept
get_slope_and_intercept <- function(X, Y, deg=1){
  zcoeff=NA
  if (deg==1){
    z = lm(Y ~ X)
    zcoeff = z$coefficients
  } else if (deg == 2) {
    X2 = X^2
    z = lm(Y ~ X + X2)
    zcoeff = z$coefficients
  }
  return(zcoeff)
}
```

```
# using the coefficients of the line provided in z
# and for each x (time) value, in argument X, it will calculate the y_hat
# and it will calculate the difference between y (actual weight) and y_hat

getSS<-function(X,Y,z){
    # using the z values, calculate the y_hat for each value
    # then calculate RMSE, sqrt ( Sum (y_hat - y)**2 / n )
    all_diffs=numeric()

    for (i in 1:length(X)){
        if (length(z) == 2){
            y_hat = X[i]*z[2] + z[1]
        } else if (length(z)==3){
            y_hat = X[i]*X[i]*z[3]+X[i]*z[2]+z[1]
        } else {
          print("something is wrong with z")
        }

        all_diffs[i]=(y_hat - Y[i])**2
    }
    diff_sum = sum(all_diffs)
    return(diff_sum)
}
```

```
# the rmse is simply the square root of the average sum of squares
getRMSE <- function(ss) {
    diff_sum = sum(ss)/length(ss)
    return(sqrt(diff_sum))

}
```

```
uniqueDiets = unique(chickweight$Diet)

ss_linear_dict = list()
ss_quadratic_dict= list()

# Linear regression of all diets
for (diet in uniqueDiets){

        df_this_diet = subset(chickweight, Diet == diet)
        print(dim(df_this_diet))
        # for each diet calculate the regression line
        X = df_this_diet$Time
        Y = df_this_diet$weight
        z = get_slope_and_intercept(X, Y, 1)

        # for each point in the diet, calculate the RMSE and add it to rmse
        # dictionary for a given diet.
        ss_linear_dict[[diet]]=numeric()
        for (i in 1:length(X)) {
          ss_linear_dict[[diet]][i]=getSS(X=X[i],Y=Y[i],z=z)
        }

  }
```

```
## [1] 192   4
## [1] 120   4
## [1] 120   4
## [1] 108   4
```

```
for (diet in uniqueDiets){

        df_this_diet = subset(chickweight, Diet == diet)
        # for each diet calculate the regression line
        X = df_this_diet$Time
        Y = df_this_diet$weight
        z = get_slope_and_intercept(X, Y, 2)

        # for each point in the diet, calculate the RMSE and add it to rmse
        # dictionary for a given diet.
        ss_quadratic_dict[[diet]]=numeric()
        for (i in 1:length(X)) {
          ss_quadratic_dict[[diet]][i]=getSS(X=X[i],Y=Y[i],z=z)
        }

  }


obs_rmse_diff=list()
for (diet in uniqueDiets){
        print(paste("======= Diet: ", diet,"=======", sep=""))
        print(paste("======= quadratic =", getRMSE(ss_quadratic_dict[[diet]]), sep=""))
        print(paste("======= linear =", getRMSE(ss_linear_dict[[diet]]), sep=""))
        obs_rmse_diff[[diet]]=getRMSE(ss_linear_dict[[diet]])-getRMSE(ss_quadratic_dict[[diet]])
```

```
        print(paste("======= difference =" ,obs_rmse_diff[[diet]], sep=""))
}
```

```
## [1] "======= Diet: 1======="
## [1] "======= quadratic =33.4824458916724"
## [1] "======= linear =33.624771261749"
## [1] "======= difference =0.14232537007662"
## [1] "======= Diet: 2======="
## [1] "======= quadratic =40.6938220465954"
## [1] "======= linear =41.0146525450394"
## [1] "======= difference =0.320830498444074"
## [1] "======= Diet: 3======="
## [1] "======= quadratic =36.0706390813104"
## [1] "======= linear =37.9034649094963"
## [1] "======= difference =1.83282582818588"
## [1] "======= Diet: 4======="
## [1] "======= quadratic =20.1679904450803"
## [1] "======= linear =20.3893457755647"
## [1] "======= difference =0.221355330484403"
```

```
# given two values, return them either in the same order
# or swapped.
randomswap<-function(specificorder){
    # x and y are two numbers
    randomorder = sample(specificorder, 2, replace=F)
    return(randomorder)
}
```

```
# For every diet,
#   repeatedly, 10,000 times,
#     for each point(a chick at a given time) in the model
#       swap the square of the error from linear and quadratic models.
#       calcuate the rmse
#     calculate the diff of rmse between the quadratic and linear models

set.seed(123)

for(diet in uniqueDiets) {
    print(paste("======= Diet: ", diet,"=======", sep=""))
    extremecount = 0
    for (j in 1:10000){
        temp_quadratic=numeric()
        temp_linear=numeric()
        for (i in 1:length(ss_quadratic_dict[[diet]])){
            randomorder = randomswap(c(ss_quadratic_dict[[diet]][i],ss_linear_dict[[diet]][i]))
            temp_quadratic[i]=randomorder[1]
            temp_linear[i]=randomorder[2]
        }
        temp_linear_rmse = getRMSE(temp_linear)
        temp_quad_rmse = getRMSE(temp_quadratic)
        rmse_diff = temp_linear_rmse - temp_quad_rmse
        if (obs_rmse_diff[[diet]] < rmse_diff) {
            extremecount = extremecount + 1
```

```
        }

    }
    print(extremecount/10000)
}
```

```
## [1] "======= Diet: 1======="
## [1] 0.2766
## [1] "======= Diet: 2======="
## [1] 0.2665
## [1] "======= Diet: 3======="
## [1] 0.0551
## [1] "======= Diet: 4======="
## [1] 0.2563
```

```
#Segregate by diet
#do 10,000 times
#  Create a bootstrap of the rows
#  Compute the RMSE for each column (linear and quadratic)
#  Record the difference of the RMSEs.

#Sort the differences and get the 90% confidence interval.

set.seed(123)

for(diet in uniqueDiets) {
    print(paste("** Diet" , diet , " **", sep=""))
    dietdf = subset(chickweight, Diet==diet)
    dietdf['sslinear']=ss_linear_dict[[diet]]
    dietdf['ssquadratic']=ss_quadratic_dict[[diet]]
    rmsediff=numeric()
    for (i in 1:10000){
        # get bootstrapped index values.
        bootstrappedindex = bootstrap(rownames(dietdf))
        sslinear_bootstrap = dietdf[bootstrappedindex,"sslinear"]
        ssquadratic_bootstrap = dietdf[bootstrappedindex,"ssquadratic"]
        rmsediff[i]=getRMSE(sslinear_bootstrap)-getRMSE(ssquadratic_bootstrap)

    }
    rmsediff = sort(rmsediff, decreasing = F)
    print(paste("lwr: ", rmsediff[500], " - upr: " , rmsediff[9500], ".", sep=""))
}
```

```
## [1] "** Diet1 **"
## [1] "lwr: -0.229188616003889 - upr: 0.502991643939502."
## [1] "** Diet2 **"
## [1] "lwr: -0.493973562065229 - upr: 1.15478807279968."
## [1] "** Diet3 **"
## [1] "lwr: -0.00656201264303746 - upr: 3.70405884545276."
## [1] "** Diet4 **"
## [1] "lwr: -0.312879156243195 - upr: 0.709456794158957."
```