R Codes for Exam

Ng Xuan Jun

25 November 2021

Contents

General Stuff

Functions for Computation of Distributions

- Note that by default, `lower.tail = TRUE`, it computes the lower tail probability or quantile
- If upper tail probability and quantiles are needed, specify `lower.tail = FALSE`

```
# Probability Density Function
d* # Add the distribution at *
dnorm()

# Cumulative Distribution Function
p* # Add the distribution at *
pnorm()

# Quantiles
q* # Add the distribution at *
qnorm()

# Generation of random samples
r* # Add the distribution at *
rnorm()
```

Distributions that are common:

- Normal Distribution: `norm`
- t - Distribution: `t`
- F - Distribution: `F`
- χ^2 Distribution: `chisq`

Formula Specification

- `x1 + x2`: $\beta_0 + \beta_1 x_1 + \beta_2 x_2$
- `-1 + x1 + x2`: $\beta_1 x_1 + \beta_2 x_2$
  - By indicating -1, we will remove the intercept term β_0
- `x1*x2`: $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \gamma_{12} x_1 x_2$
  - Taking `*` gives us all possible permutations
- `x1:x2`: $\beta_0 + \gamma_{12} x_1 x_2$
  - Taking `:` only gives us the combination term

Special Functions

`offset()` - If a variable is specified as `offset(x)` under the formula specification, the coefficient will be set as 1

`I()` - Acts as a "as.is" function whereby it will just evaluate the formula as it is

- E.g. `I(x1 * x2)` will just be `x1x2` instead of `x1 + x2 + x1x2`

Simple Linear Regression

Reading Data

```
read.csv("filename", header = )
read.table("filename", header = )
```

Plotting

```
# plot y against x
plot(x, y)

# draws the fitted regression line
# over the scatter plot of y against x
abline(lm_object)
```

Linear Model Fitting

```
# creates a linear model object
lm(y ~ x)

# get a summary of the fitted model
```

```r
summary(lm_object)

# get the anova table for the linear object
anova(lm_object)

# extract fitted values
lm_object$fitted.values

# extract residual values
lm_object$residual
```

Evaluation of Linear Model

```r
# helps to predict the values of the
# listed values
predict(lm_object, list(x_name = c(vals)),
        interval = c("none", "confidence", "prediction"),
        level = 0.9)

# gives the confidence level of the model
confint(lm_object, level = 0.9)

# gets the 95 quantile for a t distribution
# of df degrees of freedom
qt(0.95, df)

# gets cdf for a t distribution at val
# of df degrees of freedom
pt(val, df)

# if we want P(T >= val)
1 - pt(val, df)

# R Squared
summary(lm_object)$r.squared

# pearson correlation
cor()

# scatterplot of all predictors against each other
pairs()

# MSE
summary(lm_object)$sigma
```

Multiple Linear Regression

Similar to Simple Linear Regression

Computation of MCORR

```
Z <- cbind(predictors)
# sample variance of Z
v.z <- var(Z)
# sample covariance of Z and y
v.zy <- cov(Z, y)
# sample variance of y
v.y <- var(y)
mcor2 <- (t(v.zy) %*% solve(v.z) %*% v.zy)/
          v.y
```

ANOVA

Creation of Model

Note that when we are creating the model for ANOVA, we will arrange the data all in 1 continuous vector and arrange the predictor values in the same order

- Note that using `as.factor()` helps us to tackle the problem of having to create indicator (dummy) variables. If we don't then we will have to manually create the indicator variables

```
response <- c()

# manual indicator variable
u1[index_factor1] <- 1
u2[index_factor2] <- 2

lm_model <- lm(response ~ u1 + u2)

# better method
predictors <- as.factor(c())

lm_model <- lm(response ~ predictor)
```

Fitting of Main and Interaction Models

Main Models: `+` use the plus sign between predictor variables

- If the number of observations in each of the cells are **different**, note that when we read from the ANOVA table, only the SS of the last variable is correct. Therefore, if we want to get the SS of 2 variables, we will need to fit the model twice and specify the order differently. With the one that we want to get the SS at the back.
- Note that this only affects the ANOVA table and does not affect the summary table. Values for the summary table does not differ across models. Therefore we can make use of any to make inference on the summary if required

Interaction Models: `*` use the times sign between predictor variables

```r
# note the difference in the ordering
model1 <- lm(response ~ factor2 + factor1)
model2 <- lm(response ~ factor1 + factor2)

# only factor1 SS is correct
anova(model1)

# only factor 2 SS is correct
anova(model2)
```

Wald Statistics to compute F Statistics for Main Effect Model

Due to the issue with the ANOVA table under the Main Effect Model, the ANOVA table cannot be used to give us inference on the F values for the significance of the different factors. We will need to use the Wald Statistics to compute the F Statistics instead if we do not want to refit the model. Only the F Statistics for the last variable is correct.

Formula:

$$W = \beta^T \hat{\Sigma_\beta}^{-1} \beta$$

β - Coefficients of the levels of the Factor Variable (For the 1 Factor that we are comparing)

- Note that `t()` can be used to give us the transpose of β

$\hat{\Sigma_\beta}^{-1}$ - Covariance Matrix of levels of the Factor Variable

- Note the inverse of the matrix (`solve()` can be used to inverse a matrix)
- Note that we need to take out only the values that are required

F Statistics:

$$F = W/p \text{ where } F \sim F_{p,n_{ERR}}$$

p - Suppose that we are comparing Factor A with a levels, since we have a levels, $p = a - 1$ since we will have $a - 1$ regression coefficients

n_{ERR} - df of Residuals

```r
# this is only for the F Statistics of the variables that are wrong

# getting the coefficients from the main effect model
offer_main_coef <- offer_main_gender$coefficients
offer_main_vcov <- vcov(offer_main_gender) # variance covariance matrix

# note that for Wald Statistics we do not need to do adjustment

# for age_group
offer_ag_coef <- offer_main_coef[2:3]
offer_ag_vcov <- offer_main_vcov[2:3, 2:3]

# dividing by the degrees of freedom
f_stat_ag <- (t(offer_ag_coef) %*% solve(offer_ag_vcov) %*% offer_ag_coef) / 2
```

Adjustment for Main Effect Model σ^2

Note that we will need to make adjustments for the estimate of σ^2 under the Main Effect Model when we want to make inference on it.

σ^2_M - Estimate of σ^2 under the Main Effect Model

σ^2_I - Estimate of σ^2 under the Interaction Model

$\tilde{\Sigma}$ - Estimated Covariance Matrix from Main Effect Model

$p = a - 1$ - Where a is the number of levels for the factor

Adjustments:

1) Estimated Covariance Matrix: $\hat{\Sigma} = \frac{\sigma^2_I}{\sigma^2_M}\tilde{\Sigma}$

2) F Statistics: $F = \frac{\sigma^2_M}{\sigma^2_I}F_M$

- Note that the resultant F Statistics will be $F \sim F_{p,n_{\sigma^2_I}}$ instead of $F \sim F_{p,n_{\sigma^2_M}}$

3) T Statistics: $t = \frac{\sigma^2_M}{\sigma^2_I}t_M$

- Note that the resultant t Statistics will be $t \sim t_{n_{\sigma^2_I}}$ instead of $t \sim t_{n_{\sigma^2_M}}$

Computation of Overall Interaction Effect

1) Compute the value from the interaction ANOVA table (Look at the F Stats and P-value under the ANOVA table for the interaction terms)
2) Compute the Wald Statistics

Formula:

$$W = \xi^T \hat{\Sigma_\xi}^{-1} \xi$$

ξ - Coefficients of the interaction terms

- Note that `t()` can be used to give us the transpose of ξ

$\hat{\Sigma_\xi}^{-1}$ - Covariance Matrix of the interaction terms

- Note the inverse of the matrix (`solve()` can be used to inverse a matrix)
- Note that we need to take out only the values that are required

F Statistics:

$$F = W/[(a-1)(b-1)] \texttt{ where } F \sim F_{(a-1)(b-1),n-ab}$$

a : Number of levels for Factor A

b : Number of levels for Factor B

```r
# interaction model
HRC.int = lm(y~beat*time,data=HRC.dat)

b=HRC.int$coef
V=vcov(HRC.int)
# coefficients of interaction terms
b.x=b[6:9]
# covariance matrix involving the interaction terms
v.x=V[6:9,6:9]
# computation of Wald Statistics
W=t(b.x)%*%solve(v.x)%*%b.x
# Divide by (a-1)(b-1) to get the F Stats
F=W/4

# p-value
pf(F, (a-1)(b-1), n-ab, lower.tail = FALSE)
```

Computation of Inference on Particular Interaction Effects

When we want to test specific contrasts

$$T_c = \frac{c^T \xi}{\sqrt{c^T \Sigma_\xi c}} \text{ where } T_c \sim t_{n-ab}$$

c - Contrast

ξ - Coefficients of the interaction terms

- Note that `t()` can be used to give us the transpose of ξ

$\hat{\Sigma_\xi}$ - Covariance Matrix of the interaction terms

```r
# contrast matrix
c=c(1,-1,-1,1)
# test statistics
T.c = t(c)%*%b.x / sqrt( t(c)%*%v.x%*%c)
# p-value
pt(T.c,36,lower.tail=FALSE)
```

Multiple Comparison

Scheffe's Criterion

For general exploration, we will look at the estimates that we get from the linear model and see which are the contrasts that we can take that could have a big difference that is significant.

1. First way is to look at the individual estimates which in itself is the difference between the current level and the reference level. We can see that all the estimates are relatively high (no basis for how high, can just decide by yourself).

- We can look at these contrasts: $\beta_2 : \mu_2 - \mu_1$, $\beta_3 : \mu_3 - \mu_1$, $\beta_4 : \mu_4 - \mu_1$

2. Second way is to think of what are the contrasts that could be of interest to us. So for this question, we could test whether there is a difference for choosing "Yes" or "No" for both factor A and factor B

- We can look at these contrasts:
  - Main Effect of Factor B: (i.e. Difference between "Yes" and "No"): $(\mu_1 + \mu_2) - (\mu_3 + \mu_4) = \beta_2 - \beta_3 - \beta_4$
  - Main Effect of Factor A: (i.e. Difference between "Yes" and "No"): $(\mu_1 + \mu_3) - (\mu_2 + \mu_4) = \beta_3 - \beta_2 - \beta_4$

We will make use of the p-value computation using the Scheffe's Criterion which is given by $P(F_{a-1,n_{ERR}} \geq \frac{T_C^2}{a-1})$ to control the overall type 1 error rate.

```
# testing of the contrasts
contrast_matrix <- matrix(c(1, 0, 0,
                            0, 1, 0,
                            0, 0, 1,
                            1, -1, -1,
                            -1, 1, -1),
                          ncol = 3, byrow = TRUE)

# dropping the intercept term
clinical_combined_coef <- clinical_combined$coefficients[-1]
# variance matrix (note that we will drop the intercept term here too)
clinical_combined_vcov <- vcov(clinical_combined)[-1, -1]

# estimated c
clinical_combined_c <- contrast_matrix %*% clinical_combined_coef
# standard error of c
clinical_combined_se <- sqrt(diag(contrast_matrix %*% clinical_combined_vcov %*% t(contrast_matrix)))
# t_value of c
clinical_t_val <- clinical_combined_c / clinical_combined_se
# p-value of c
clinical_p_val <- pf(clinical_t_val ^ 2 / 3, 3, 41, lower.tail = FALSE)

# computing the scheffe's Criterion
clinical_scheffe <- sqrt((4 - 1) * qf(0.05, 3, 41, lower.tail = FALSE))

data.frame("T-Values" = clinical_t_val, "P-Values" = clinical_p_val)
```

Tukey's Criterion

- Need to use the Q table to compute the values

Bonferroni Criterion

- Take the p-values and multiply by the number of variables that we are testing or divide the level of significance by the number of variables

- Note that when we compute the p values, we will need to multiply by 2 because we are testing for 2 sided test

```
# contrast
c1=c(0.5,-0.5,-0.5); c2=c(-0.5,0.5,-0.5)
L1=t(c1)%*%b; s1=sqrt(t(c1)%*%V%*%c1); T1= L1/s1
L2=t(c2)%*%b; s2=sqrt(t(c2)%*%V%*%c2); T2= L2/s2
# test statistics
T=c(T1,T2)
# p-values need to multiply by 2 and also the number of variables
# note that it is the abs() value
2*2*pt(abs(T),15,lower.tail=FALSE)
```

Variable Selection: Criteria

Note that if we require R^2 or R_a^2 we should use `lm` model instead

Note that we should fit a `glm` object instead to compute the criteria

- Note that the CV Score for k folds is taking random samples so from each run, we may get different results

```
glm(response ~ predictors)

library(stats)
# compute AIC
AIC(object, k)
# compute BIC
BIC(object)

# compute CV Score
library(boot) # remember to load the library
# 1 fold CV
cv.glm(data, glm_model, K = 1)$delta[2]
# k fold CV
cv.glm(data, glm_model, K = k)$delta[2]
```

Variable Selection: Procedures

Sequential Procedures

Note that we need to load the ***MASS*** *package for the selection procedures*

To use the `stepAIC` function:

1. Supply the empty/full `lm` model

```
* Note that the empty model does not contain any variables

* Note taht the full model contains all the variables
```

2. State the `scope` which is passed as a list including the lower and upper bound for the parameters that should be considered for the final model

- `lower` - Supply a formula for the lower bound
- `upper` - Supply a formula for the upper bound

3. State the `direction` of the sequential selection procedure

- `forward` - Forward selection, make sure that an empty model is supplied
- `backward` - Backward selection, make sure that a full model is supplied
- `both` - Upward Selection, make sure that empty model is supplied. Downward Selection, make sure that full model is supplied

```
library(MASS)

empty_model <- lm(y ~ 1)
full_model <- lm(y ~ x1 + x2 + x3 + x4 + x5)

model <- stepAIC(empty_model/full_model,
                 scope = list(lower = ~ 1,
                              upper = ~ x1 + x2 + x3 + x4 + x5),
                 direction = "both"/"forward"/"backward")
```

LASSO

Note that we need to load the following packages

```
library(iterators)
```

```
library(foreach)
```

```
library(Matrix)
```

```
library(shape)
```

```
library(glmnet)
```

When we use `glmnet()`, we will need to pass in the predictor variables `x` as a matrix and the response variable `y` as a vector

- Note that `glmnet()` does not take in factor variables. So we need to create dummy variables if we want to supply the factor variables into the model

```
library(iterators)
library(foreach)
library(Matrix)
library(shape)
library(glmnet)


# preparation of predictor variables
predictors <- as.matrix(predictor_values)
```

```
response

# fitted model
model_fit <- cv.glmnet(x = predictors, y = response)
# lambda value that gives smallest value
model_fit$lambda.min

# coefficients of using the minimum lambda value
# find the values that have non-zero coefficient and they will form final model
coef(model_fit, s = model_fit$lambda.min)
```

Model Diagnostics

Raw Materials Computation

- Note to use `glm` model to fit the model for the computation of the raw materials

```
# fitting the model and computing the raw materials
fitted_model <- glm(y ~ x1 + x2 + x5 + x8 + x9 + x10,
                    data = smsa_data) # note to use glm

# fitted values
yhat <- fitted_model$fitted.values
# pearson residuals
residuals <- residuals(fitted_model, type = "pearson")
# hat values
hat_values <- hatvalues(fitted_model, type = "diagonal")
# influence (used to computation of remaining 3 things)
infl = influence(fitted_model, do.coef = FALSE)
# standardised residuals
resi_sta <- rstandard(fitted_model, infl, type = "pearson")
# studentised deleted residuals
resi_stu <- rstudent(fitted_model, infl, type = "pearson")
# cook's distance
cook <- cooks.distance(fitted_model, infl, res = infl$pear.res,
                       disperson = summary(fitted_model)$disperson,
                       hat = infl$hat)
```

We note that for every round of diagnostics that we do, we should check the following 3 things:

1. **Non-Linearity**: Check whether the linear terms of the predictor variables are appropriate by the residual plots

```
par(mfrow=c(2,3))
plot(x1, residuals,xlab="Residual vs. x1")
plot(x2, residuals,xlab="Residual vs. x2")
plot(x5, residuals,xlab="Residual vs. x5")
plot(x8, residuals,xlab="Residual vs. x8")
plot(x9, residuals,xlab="Residual vs. x9")
plot(yhat,residuals,xlab="Residual vs. fitted value")
```

2. **Homogeneity**: Checking if the variance is constant through the data

```
plot(yhat, residuals,xlab="Residual vs. fitted values")
```

3. **Normality**: Check if the residuals follow a normal distribution

```
qqnorm(resi_sta,xlab="Q-Q plot of standardized residuals")
```

4. **Outliers**: Check for Outliers

Note that we can test the significance of outliers one by one by identifying one, test using dummy variables, remove from the data, continue until no more significant outliers

We can also test them all at once by creating dummy variables all at once

```
par(mfrow=c(2,2))
# checking leverage
qqnorm(h,xlab="",ylab="",main="Q-Q  plot of the hat values")
# checking consistency
qqnorm(rstu,xlab="",ylab="",main="Q-Q  plot of the studentized deletion residuals")
# checking influence (most important)
qqnorm(cook,xlab="",ylab="",main="Q-Q  plot of the Cook's distances")

# ordering the values
n = length(cook)
cbind( order(cook), cook[order(cook)],
       order(rstu), rstu[order(rstu)],
       order(h),h[order(h)] )[(n-5):n,]

# test the significance of observation 1
u=rep(0,n); u[1]=1;
f.outlier = lm(y~x1 + x2 + x5 + x8 + x9 + x10+u,data=p.dat)
summary(f.outlier)

# testing all at once
n = length(cook)
u1=rep(0,n); u2=u1; u3=u1; u4=u1; u5=u1; u6=u1
u1[1]=1; u2[3]=1;u3[4]=1; u4[5]=1;u5[7]=1;u6[9]=1
outlier =  lm(y~x1 + x2 + x5 + x8 + x9 + x10+u1+u2+u3+u4+u5+u6,data=p.dat)
summary(outlier)
```

Weighted LSE & Multicollinearity

Weighted LSE

Once we find the weights, we can supply it to the `lm` model under `weight`, we supply the weights

Note that we need to multiply the residuals and fitted values with `sqrt(w)` when we want to compare the residuals and fitted values to see if the variance is constant afterwards

Note that if we take the variance as a function of mean. We can take the weight as the inverse of the exponential of the fitted values of the log model.

```
## unweighted fitinng
fit0=lm(y~T+G+S) #unweighted
r0=fit0$resid; y0=fit0$fitted
plot(y0,r0)

## Fitting with weight n --- constant variance for individuals
w1=n
fit1=lm(y~T+G+S,weight=w1)
r1=fit1$resid; y1=fit1$fitted
r1.w=r1*sqrt(w1); y1.w =y1*sqrt(w1)
plot(y1.w,r1.w,xlab="Residual plots with weight =n")


## Fitting with weight n/s^2, non-constant variances for groups
w2=n/s^2
fit2=lm(y~T+G+S,weight=w2)
r2=fit2$resid; y2=fit2$fitted
r2.w=r2*sqrt(w2); y2.w =y2*sqrt(w2)
plot(y2.w,r2.w,xlab="Residual plots with weight =n/s^2")

## Fitting with functional estimate of weights

plot(y,s)
w.fit = lm(log(s)~log(y))
log.s =w.fit$fitted; s3 =exp(log.s)
w3=n/s3^2   #group indivicual variance as function of mean
fit3=lm(y~T+G+S,weight=w3)
r3=fit3$resid; y3=fit3$fitted
r3.w=r3*sqrt(w3); y3.w =y3*sqrt(w3)
plot(y3.w,r3.w,xlab="Residual plots with functional estimated weight")
```

Multicollinearity

- Plot Scatterplot to see if there is linear trend between the variables using `pairs()`
- Compute correlation to see if there is strong correlation between variables using `cor()`
- Check the difference in coefficients after removing certain variables

VIF

- Compute VIF (Use Prof Function). Supply a linear model to it
  - Remove the one that has the highest VIF above 5 and repeat the process by refitting model and computing VIF until there is no more VIF more than 5

```
vifChen = function(object) {
 ## Input:
  # object -- a fitted lm object.
 ## Output:
  # vif -- variance inflation factors
```

```
 X = object$x[,-1]
 V = vcov(object)[-1,-1]
 n = dim(X)[1]
 sigma = summary(object)$sigma

 v = diag(V)
 S = diag(var(X))*(n - 1)
 vif = v * S / sigma^2
 vif
}
```

Ridge Regression

Load the `MASS` package

```
library(MASS)

# generate lambda values
lambda <- seq(1, 50, 0.01)
ridge_model <- lm.ridge(formula, data, lambda)

# get GCV values
CV <- ridge_model$GCV
# get lambda values used
lambda_val <- ridge_model$lambda
# find the lambda with lowest GCV
lambda.best <- lambda_val[order(CV)[1]]
```

Variable Transformation

Variance Stabilisation Transformation

1) Proportion Response: $\arcsin(\sqrt{x})$

```
org.fit=lm(y~g)
r.o=org.fit$resid
y.o = org.fit$fitted

y.t=asin(sqrt(y))
tra.fit=lm(y.t~g)
r.t=tra.fit$resid
y.t = tra.fit$fitted

par(mfrow=c(1,2))
plot(y.o,r.o,main="Original fitting")
plot(y.t,r.t,main="Transformed fitting")

summary(org.fit)
summary(tra.fit)
```

2) Count Response: $\sqrt{x}$

Box-Cox Transformation

Determination of α

1) Method 1: Fit the regression model for the residuals against mean and find the relationship between the residuals and mean. Take α as nearest estimate of β_1

```
# ungrouped
alpha.fit1=lm(log(abs(r.o))~log(y.o))
summary(alpha.fit1)

# grouped
YY = matrix(y,ncol=4,byrow=T)
ybar = apply(YY,1,mean)
sd = sqrt(apply(YY,1, var))
alpha.fit2 = lm(log(sd)~log(ybar))
summary(alpha.fit2)
```

2) Method 2: For grouped only. Select a few values of α and for each of them compute the max vs min of the ratio of $\frac{s_i}{\bar{y}_i^{\alpha}}$. Select the one with the smallest value

```
## method II
alpha=c(-1, -0.5,0.5,1,1.5,2,3)
R=NULL
for (i in 1:length(alpha) ) {
  R[i] = max(sd/ybar^alpha[i])/ min(sd/ybar^alpha[i])
}
```

Determination of λ

1) Grouped: Select a few values of λ compute the ratio of max and min of s_λ when we give the transformation of y^λ. Select the value of λ such that the ratio is closet to 1

```
lambda=c(-1,-0.5,0.001,0.5); s1=NULL; s2=NULL
for (i in 1:4) {
  z1=(y1^lambda[i]-1)/lambda[i]; z2=(y2^lambda[i]-1)/lambda[i]
  s1[i]=sqrt(var(z1)); s2[i]=sqrt(var(z2))
}
s.ratio = s1/s2
```

2) Ungrouped: Select a few values of λ, give the transformation of y^λ. Select the value of λ such that the residual plots exhibit a random pattern