Mathematical Statistics II

Statistical Computing Activity: Module 7

One purpose of these Statistical Computing Activities (SCAs) is to give you a chance to explore statistics using the computer. Another purpose is to give you more skills in thinking about the randomness that is life.

Usually, like here, these SCAs will have a theme and several problems dealing with that theme or purpose. The reason for that extra layer of complexity is to tie what we do in the class with what we can use these techniques for in our lives as statisticians and/or consultants and/or full members of a democratic society.

The purpose of this activity is to give you some practice in collecting and analyzing data. Since the data consist of one numeric variable and one categorical variable, we will use one-way analysis of variance to perform the analysis. Remember that there is nothing new in analysis of variance that was not found in linear regression. The only difference is that regression focuses on the effects of the levels and ANOVA focuses on the effects of the variables.

⋆⋆⋆

The Procedure

The research question is "Which of these three programs is fastest?"

Step One: Collect Data. One of many important considerations in selecting the 'optimal' statistical program is speed. Here, we will compare the speed of Mathematica, Python, and R in generating a million random variates from a standard Normal distribution.

To test this, each of the three of you will select one of the three languages above. You will run the associated code below fifteen times ($n_i = 15$), recording the elapsed time in each run.

Here are the codes for the three languages:

Mathematica.

```
start = DateList[]
x = RandomVariate[NormalDistribution[], 10^6]
end = DateList[]

end - start
```

Python.

```
import numpy as np
import datetime as dt

start = dt.datetime.utcnow()
x=np.random.normal(0,1, 1000000)
end = dt.datetime.utcnow()

print(end-start)
```

R.

```
start=Sys.time()
x=rnorm(1e6)
end=Sys.time()

end-start
```

Remember to record the elapsed time after each run. That will serve as your data. Write your results on the front board, because we will all be working with the same set of data.

Step 2: Get the data into R. Now that everyone has their data on the front board, get that data into R. Here is how I would start:

```
m = "Mathematica"
p = "Python"
r = "R"
program = c( rep(m,15), rep(p,15), rep(r,15) )

mTime = c(
pTime = c(
rTime = c(
time  = c( mTime, pTime, rTime )
```

As in our notes, the next step is to create the X and Y matrices. Since this is one-way ANOVA, the design and response matrices are rather straight-forward to create using some R functions.

```
I3 = diag(3)
J15 = matrix(rep(1,15), ncol=1)
I3J15 = I3 %x% J15
X = matrix( c(rep(1,45),I3J15), ncol=4)
Y = matrix(time, ncol=1)
```

Step 3: Hard Analysis. In this part, you will do the OLS calculations using matrix multiplication. In R, the "multiply matrix" function is `%*%`. The inverse is `solve()`. The transpose function is `t()`. Thus, the **b** vector would be:

```
b = solve(t(X) %*% X) %*% t(X) %*% Y
```

Arrrrgh!!! What happened?????

Oh yeah, I forgot the multicollinearity issue. =) So, to fix that, we just need to drop a row from the design matrix. Whichever row we drop will serve as the base level. All estimated effects will be measured *with respect to* that variable.

The column we drop is usually determined by the science question. So, each of you needs to drop the column corresponding to your program. That means you estimates are measured *with respect to* your program.

For instance, to drop the first column from the design matrix (the column of 1s corresponding to the μ), run

```
X = X[ ,-1]
```

That line will drop the common mean design information in X, ensuring that all effects are estimated with respect to zero (the estimates will be the sample means). Remember to drop yours.

Problem 4: First Calculations

Calculate the following matrix:

a) **b**

Calculate

b) the sample means in each group

Take time to make sure that you see the connection between the sample means and the **b** vector. The connection should be evident.

Step 5: The SS Values. Now, let us determine the SS values. If you set up things as in the example code above, this should be rather straight-forward (if not easy). Remember that if the design is balanced ($n_i = n, \forall i$), then

$$\begin{aligned} SST &= \sum_{i=1}^{k} \sum_{j=1}^{n_i} (\bar{y}_i - \bar{y})^2 \\ &= \sum_{i=1}^{k} n(\bar{y}_i - \bar{y})^2 = n \sum_{i=1}^{k} (\bar{y}_i - \bar{y})^2 \end{aligned}$$

In R, we can get this through this line

```
SST = 15*(mean(mTime)-mean(time))^2 +
      15*(mean(pTime)-mean(time))^2 +
      15*(mean(rTime)-mean(time))^2
```

Similarly, the other two SS terms are

```
SSE = sum( (mTime-mean(mTime))^2 ) +
      sum( (pTime-mean(pTime))^2 ) +
      sum( (rTime-mean(rTime))^2 )
```

and

```
TSS = sum( (time-mean(time))^2 )
```

The other parts of the ANOVA table are

```
# df
Tdf = dim(X)[1]-1
dfT = dim(X)[2]-1
dfE = Tdf-dfT

# MS
MST = SST/dfT
MSE = SSE/dfE

# F
F = MST/MSE

# p-value
pf(F, df1=dfT,df2=dfE, lower.tail=FALSE)
```

Problem 6: The ANOVA Table
From those values, complete the ANOVA table.

Step 7: The Easy Analysis. In this section, we will use R to perform these calculations for us. It all comes down to the aov function. Run the following lines:

```
mod = aov(times~program)
summary(mod)
```

Problem 8: The ANOVA Table
From those values, complete the ANOVA table based on the R output.

Step 9: Check Yourself. Make sure that the two ANOVA tables are identical.

Step 10: Multiple Comparisons. In the previous steps, especially from the calculated p-value, you determined that the null hypothesis (that the population means are all equal) does not match reality. In other words, you determined that "at least one population mean differs from the others."

That is like eating a half of a Uno bar. Cool, awesome, and tasty — but ultimately unsatisfying. What we would really like to determine is Which is different and in which way? More specifically, we would like to know which is fastest.

That requires us to perform post-hoc multiple comparisons. There are many, many, many post-hoc tests. (Create one and get a Ph.D.!)

Each has a strength and a weakness. The Fisher LSD test is the most conceptually clear, followed by the Bonferroni adjustment and the Tukey HSD test. Others may be improvements over these simple tests, but they are all more difficult to calculate by hand.

There are many, many R functions available to perform these multiple comparisons. Almost all require loading different packages. Popular packages include the multcomp package and the agricolae package. In base R, there is the `TukeyHSD` function that performs Tukey's Honestly Significant Difference (HSD) test. As a general test, it is not dominated by any other. As such, without additional knowledge of the problem, it is a good basic test. Our textbook covers how to perform this test using our calculator and fingers. Here is how we do it in R:

```
TukeyHSD(mod)
```

Problem 11: The Final Ordering

So, look carefully through the output of that function... there is a lot! By this point, with a little thought, you should be able to interpret the output and determine which of the programs is (statistically) significantly different from the others.

- So, based on the results of Tukey's HSD test, statistically order the results from your analysis.