Statistical Inference Project # 2

Liam Flaherty

Professor Post

NCSU: ST502-002

April 17, 2024

Contents

1 Introduction **3**

2 Theoretical Preliminaries **4**
Degrees Of Freedom 4
Likelihood Derivations 4
Pearson's Chi-Squared Test 6
Equivalence Of McNemar's Test 7

3 Hypothesis Test **8**

4 Simulation **9**
Alpha Control 11
Power Analysis 13

5 Full R Code **14**

6 References **17**

1 Introduction

McNemar's Test is used to judge the symmetry of a two-by-two contingency table [1]. In applications, this can look like testing the difference in concordant/discordant pairs. An important rationale for why one might use McNemar's Test as opposed to some other test is that the observed pairs are not independent, since each observation is self-dependent.

In this project we are given data on 250 individuals who undergo treatment for an illness. Half of the individuals are randomly assigned to take drug A the first time their symptoms flare up, and then assigned to take drug B the second time their symptoms flare up. The other half of individuals are assigned to do the reverse (first take drug B, then drug A). Data was collected on each subject after each episode to determine if the administered drugs were successful in treating the symptoms or not. In total, 85 people experienced relief with both drugs, 40 people experienced relief with drug A only, 15 people experienced relief with drug B only, and 110 people didn't experience relief with either drug.

Of interest is whether the drugs have a different probability of giving relief. Our null hypothesis is that there is no relationships between the drugs and the probability of relief. An easy result (Proposition 2.0.1, Page 4) shows that this hypothesis essentially says that the probability one gets better with drug A but not drug B is equal to the probability one gets better with drug B but not drug A. Our alternative hypothesis is that all the probabilities are free (subject to the constraint they sum to one).

Label $n_{1,2}$ as the number of people who improve with drug B but not drug A, $n_{2,1}$ as the number of people who improve with drug A but not drug B, and $n_{2,2}$ as the number of people who don't improve with either. Further, label $\dim(\Omega)$ as the number of free parameters over the whole parameter space, and $\dim(\omega_0)$ as the number of free parameters over the null parameter space. McNemar's test statistic is $\frac{(n_{1,2}-n_{2,1})^2}{n_{1,2}+n_{2,1}}$ (Proposition 2.0.6, Page 7), which we claim without proof converges in distribution to a chi-squared random variable with $\dim(\Omega) - \dim(\omega_0) = 1$ degree of freedom (Proposition 2.0.2, Page 4). We reject our null hypothesis for large observed values of this reference distribution (where "large" is of course a function of our significance level).

In Section 3, we show the results and derivation of our hypothesis test. In particular, we record a p-value for our goodness of fit of about 0.0007 and reject the null hypothesis of "equal probability of relief" after observing a test statistic of about 11.364 (well inside our rejection region of > 3.84).

In Section 4, we perform simulation for our test with the goal of conducting a power-analysis for certain alternatives, and determining how well our tests do at controlling Type I Error. In doing so, we reach some natural conclusions— for example, our false positive rate should approach our significance level after many observations, and the power of a test moves in tandem with sample size and the difference in true means.

2 Theoretical Preliminaries

Proposition 2.0.1. In a two-by-two table of probabilities given by concordant/discordant pairs, the statement that both $p_{1,\bullet} = p_{\bullet,1}$ and $p_{2,\bullet} = p_{\bullet,2}$ is equivalent to the statement $p_{1,2} = p_{2,1}$.

Proof. This is just arithmetic. First assume that both $p_{1,\bullet} = p_{\bullet,1}$ and $p_{2,\bullet} = p_{\bullet,2}$. Then since $p_{1,\bullet} = p_{1,1} + p_{1,2}$ and since $p_{\bullet,1} = p_{1,1} + p_{2,1}$, by our assumption, $p_{1,1} + p_{1,2} = p_{1,1} + p_{2,1}$ and thus $p_{1,2} = p_{2,1}$.

Now assume that $p_{1,2} = p_{2,1}$. Then $[p_{\bullet,1} = p_{1,1} + p_{2,1}] = [p_{1,1} + p_{1,2} = p_{1,\bullet}]$ and further $[p_{\bullet,2} = p_{1,2} + p_{2,2}] = [p_{1,1} + p_{2,1} = p_{1,1} + p_{1,2} = p_{2,\bullet}]$. This proves the equivalence. ■

Proposition 2.0.2. In the problem outlined in the introduction, the degrees of freedom for our asymptotically equivalent test statistic is 1.

Proof. The full parameter space has dimension three, since there are four options with a sum to one constraint; $\dim(\Omega) = 4 - 1 = 3$. The null parameter space has dimension two, since the null hypothesis equates two pairs of probabilities, and since there is still a sum to one constraint; $\dim(\omega_0) = 4 - 1 - 1 = 2$. Together, we have $\dim(\Omega) - \dim(\omega_0) = 1$. ■

Proposition 2.0.3. In a multinomial distribution with m classes where y_i denotes the observed number of objects in class i, the maximum likelihood estimator of the parameter p_i is $\underset{\sim}{p}_{i_{\text{MLE}}} = \frac{y_i}{n}$.

Proof. The likelihood function is given by $L(p_1, \ldots, p_m) = \frac{n!}{p_1! \cdots p_m!} \prod_{i=1}^{m} p_i^{y_i} = n! \prod_{i=1}^{m} \frac{p_i^{y_i}}{y_i!}$.

The log-likelihood is then $l(p_1, \ldots, p_m) = \ln(n!) + \sum_{i=1}^{m} y_i \ln(p_i) - \sum_{i=1}^{n} \ln(y_i!)$

With the introduction of our sum-to-one constraint, the Lagrangian is:

$$\mathcal{L}(p_1, \ldots, p_m) = l(p_1, \ldots, p_m) + \lambda(1 - \sum_{i=1}^{n} p_i)$$

Finding the maximum: $\frac{\partial}{\partial p_i}\mathcal{L}(p_1, \ldots, p_m) = y_i \frac{1}{p_i} - \lambda \implies p_i = \frac{y_i}{\lambda}$

Plugging these values into our constraint: $\frac{1}{\lambda} \sum_{i=1}^{m} y_i = 1 \implies \frac{n}{\lambda} = 1 \implies \lambda = n$

Re-substituting back to our derived maximum, we reach our conclusion: $\underset{\sim}{p}_{i_{\text{MLE}}} = \frac{Y_i}{n}$.

■

Proposition 2.0.4. In a restricted multinomial distribution with 4 classes (two of which are identical) where y_i denotes the observed number of objects in class i, the maximum likelihood estimator of the restricted parameters (call them p_2 and p_3) is $\underset{\sim}{p_{i_{\text{Null}}}} = \frac{y_2+y_3}{2n}$. The unrestricted parameters have a MLE of $\underset{\sim}{p_{1_{\text{Null}}}} = \frac{y_1}{n}$ and $\underset{\sim}{p_{4_{\text{Null}}}} = \frac{y_4}{n}$ respectively.

Proof. We use the same set-up as Proposition 2.0.3 while considering the additional restriction $p_2 = p_3$. With these two constraints, our Lagrangian becomes (where c is some constant unrelated to our parameters of interest):

$$\mathcal{L}(p_1, \ldots, p_4) = c + \sum_{i=1}^{4} y_i \ln(p_i) + \lambda_1(1 - \sum_{i=1}^{4} p_i) + \lambda_2(p_2 - p_3)$$

Taking the partials, we see:

$$\frac{\partial}{\partial p_1}\mathcal{L}(p_1, \ldots, p_4) = \frac{y_1}{p_1} - \lambda_1 \qquad \frac{\partial}{\partial p_4}\mathcal{L}(p_1, \ldots, p_4) = \frac{y_4}{p_4} - \lambda_1$$
$$\frac{\partial}{\partial p_2}\mathcal{L}(p_1, \ldots, p_4) = \frac{y_2}{p_2} - \lambda_1 + \lambda_2 \qquad \frac{\partial}{\partial p_3}\mathcal{L}(p_1, \ldots, p_4) = \frac{y_3}{p_3} - \lambda_1 - \lambda_2$$

After setting these all to zero, writing each equation in terms of y_i, and then recognizing that $\sum_{i=1}^{4} y_i = n$, we can write:

$$\begin{aligned} n &= \lambda_1 p_1 + (\lambda_1 - \lambda_2)p_2 + (\lambda_1 + \lambda_2)p_3 + \lambda_1 p_4 \\ &= (p_1 + p_2 + p_3 + p_4)\lambda_1 + (p_2 - p_3)\lambda_2 && \text{Grouping} \\ &= ([1])\lambda + ([0])\lambda_2 = \lambda_1 && \text{Constraints} \end{aligned}$$

Resubstituting to our equations for y_1 and y_4, we get our MLEs for those two:

$$\frac{y_1}{p_1} = n \implies \underset{\sim}{p_{1_{\text{Null}}}} = \frac{y_1}{n} \text{ and } \frac{y_4}{p_4} = n \implies \underset{\sim}{p_{4_{\text{Null}}}} = \frac{y_4}{n}$$

Then using the constraint $\sum_{i=1}^{4} p_i = 1$ and plugging in the above, we see:

$$\begin{aligned} 1 &= \frac{y_1}{n} + \frac{y_4}{n} + p_2 + p_3 = \frac{y_1}{n} + \frac{y_4}{n} + 2p_2 && \text{Constraint } p_2 = p_3 \\ n &= y_1 + y_4 + 2p_2 n \end{aligned}$$

$$y_2 + y_3 = 2np_2 \implies \frac{y_2 + y_3}{2n} = \underset{\sim}{p_{2_{\text{Null}}}} = \underset{\sim}{p_{3_{\text{Null}}}} \qquad \text{Constraint } n = \sum_{i=1}^{4} y_i = n$$

This proves our result. ∎

Proposition 2.0.5. The likelihood ratio test statistic when conducting a goodness of fit test for the multinomial distribution (with m classes) against a restricted model (with $k-1$ free parameters) is $2\sum_{i=1}^{m} Obs_i \ln\left(\frac{Obs_i}{Exp_i}\right)$, which asymptotically follows a χ^2_{m-k-1}

Proof. Where $\Theta = (p_1, \ldots, p_m)$, our likelihood ratio is: $\Lambda = \frac{\max\limits_{\Theta\in\omega_0} L(\Theta)}{\max\limits_{\Theta\in\Omega} L(\Theta)} = \frac{L(\underset{\sim}{\Theta}_{\text{Null}})}{L(\underset{\sim}{\Theta}_{\text{MLE}})}$.

From the above proposition (Proposition 2.0.3, Page 4), we know we will get cancellation between the likelihood functions and so are left with $\Lambda = \frac{\underset{\sim}{p_1}_{\text{Null}}{}^{y_1}\cdots\underset{\sim}{p_m}_{\text{Null}}{}^{y_m}}{\underset{\sim}{p_1}_{\text{MLE}}{}^{y_1}\cdots\underset{\sim}{p_m}_{\text{MLE}}{}^{y_m}} = \prod_{i=1}^{m}\left(\frac{\underset{\sim}{p_{i_{\text{Null}}}}}{\underset{\sim}{p_{i_{\text{MLE}}}}}\right)^{y_i}$.

From large-sample theory, we then have:

$$
\begin{aligned}
-2\ln(\Lambda) &\overset{H_o}{\underset{\bullet}{\sim}} \chi^2_{m-k-1} \\
-2\sum_{i=1}^{m} y_i \ln\left(\frac{\underset{\sim}{p_{i_{\text{Null}}}}}{\underset{\sim}{p_{i_{\text{MLE}}}}}\right) &\overset{H_o}{\underset{\bullet}{\sim}} \chi^2_{m-k-1} \\
2\sum_{i=1}^{m} y_i \ln\left(\frac{\underset{\sim}{p_{i_{\text{MLE}}}}}{\underset{\sim}{p_{i_{\text{Null}}}}}\right) &\overset{H_o}{\underset{\bullet}{\sim}} \chi^2_{m-k-1} \\
2\sum_{i=1}^{m} y_i \ln\left(\frac{y_i}{n\underset{\sim}{p_{i_{\text{Null}}}}}\right) &\overset{H_o}{\underset{\bullet}{\sim}} \chi^2_{m-k-1} \qquad \text{From Proposition 2.0.3, Page 4}
\end{aligned}
$$

Under the null hypothesis, the expected number of observations in class i is precisely $n\underset{\sim}{p_{i_{\text{Null}}}}$ (since the expected value of a Binom(n,p) is np). Then since y_i are our observed values, we can write the above as $2\sum_{i=1}^{m} Obs_i \ln\left(\frac{Obs_i}{Exp_i}\right) \overset{H_o}{\underset{\bullet}{\sim}} \chi^2_{m-k-1}$ ■

Example: Applying the above derivation to the problem outlined in the introduction, we break up the sum of our four classes to first sum over two rows, and then sum over the two columns. Proposition 2.0.2 showed that our degrees of freedom is 1, so we are left with $2\sum_{j=1}^{2}\sum_{i=1}^{2} Obs_{i,j} \ln\left(\frac{Obs_{i,j}}{Exp_{i,j}}\right) \overset{H_o}{\underset{\bullet}{\sim}} \chi^2_1$.

Proposition 2.0.6. We showed that $2\sum_{j=1}^{2}\sum_{i=1}^{2} Obs_{i,j} \ln\left(\frac{Obs_{i,j}}{Exp_{i,j}}\right) \overset{H_o}{\underset{\sim}{\bullet}} \chi^2_1$ in the example to Proposition 2.0.5 above. We claim without proof that this is asymptotically the same as Pearson's Chi-Square statistic, $\sum_{i=1}^{m} \frac{(Obs_i - Exp_i)^2}{Exp_i}$. For the problem outlined in the introduction, Pearson's Chi-Square statistic can be written as $\sum_{j=1}^{2}\sum_{i=1}^{2} \frac{(Obs_{i,j} - Exp_{i,j})^2}{Exp_{i,j}}$, which we claim is the same thing as $\frac{(n_{1,2}-n_{2,1})^2}{n_{1,2}+n_{2,1}}$ (or equivalently $\frac{(y_3-y_2)^2}{y_3+y_2}$ if we use the same notation as above, where y_i indicates the observed number of objects in class i).

Proof. From Proposition 2.0.4, we know that $p_{1,1_{\text{Null}}} = \frac{y_1}{n}$ and $p_{2,2_{\text{Null}}} = \frac{y_4}{n}$. Then since $Exp_{i,j}$ is $n \cdot p_{i,j_{\text{Null}}}$, $Exp_{1,1} = y_1$ and $Exp_{2,2} = y_4$. Thus the numerator in Pearson's test for those cases is $\underset{\sim}{(Obs_i - Exp_i)^2} = (y_i - y_i)^2 = 0$; the only contribution to the statistic comes from the discordant cells.

Again from Proposition 2.0.4, we know that $p_{1,2_{\text{Null}}} = p_{2,1_{\text{Null}}} = \frac{y_2+y_3}{2n}$. Then since $Exp_{i,j} = n \cdot p_{i,j_{\text{Null}}}$, we can write $Exp_{1,2} = Exp_{2,1} = n\frac{y_2+y_3}{2n} \underset{\sim}{=} \frac{y_2+y_3}{2}$. Plugging in these expectations, Pearson's test statistic can be written:

$$\frac{\left(y_2 - \frac{y_2+y_3}{2}\right)^2}{\frac{y_2+y_3}{2}} + \frac{\left(y_3 - \frac{y_2+y_3}{2}\right)^2}{\frac{y_2+y_3}{2}} = \frac{\left(\frac{y_2-y_3}{2}\right)^2}{\frac{y_2+y_3}{2}} + \frac{\left(\frac{y_3-y_2}{2}\right)^2}{\frac{y_2+y_3}{2}} = \frac{\frac{2(y_3-y_2)^2}{4}}{\frac{y_2+y_3}{2}} = \frac{(y_3-y_2)^2}{y_3+y_2} = \frac{(n_{1,2}-n_{2,1})^2}{n_{1,2}+n_{2,1}}$$

This proves our result. ■

3 Hypothesis Test

Our null hypothesis was that there was no relationship between the drugs and relief: $H_0 : p_{1,2} = p_{2,1}$. Our alternative hypothesis was that the cells were free. Where $n_{1,2}$ represents the observed number of people who improved with drug B but not drug A, and where $n_{2,1}$ represents the observed number of people who improved with drug A but not drug B, our test statistic is $\frac{(n_{1,2}-n_{2,1})^2}{n_{1,2}+n_{2,1}}$ (Proposition 2.0.6, Page 7). Asymptotically, this test statistic follows a χ^2_1 (Proposition 2.0.3, Page 4), which we choose to use for our distribution under the null hypothesis.

We reject the null hypothesis in favor of the alternative for large values of our test stat. We set up our rejection threshold with an arbitrarily chosen significance level of 5% (i.e. when the null hypothesis is true, we should only incorrectly reject the null hypothesis 5% of the time). The below R code shows how we calculated the rejection regions and p-values. Of note is a recorded p-value of about 0.0007, which we verify with the mcnemar.test() function in R's "stat" package. Since our observed test statistic (11.364) lies inside our rejection region (> 3.84), we reject our null hypothesis.

```
###1. Load Required Packages And Data###
install.packages("stats")
library(stats)

both=85; neither=110; a=40; b=15                    #e.g. 15 got better with B but not A#
mydata=c(both, b, a, neither)                       #Put data into a vector#
mydf=data.frame(c(both, a), c(b, neither))          #Put data into data frame#
mymatrix=matrix(mydata, nrow=2, ncol=2, byrow=TRUE) #Reluctantly put data into a matrix#



###2. Hypothesis Test Set-up###
teststat=function(discordant1, discordant2) {
  ((discordant1-discordant2)^2/(discordant1+discordant2))    #Pearson's Chi-Square#
}

rr_chi=function(sig, df) {                          #Reject for large#
  qchisq(1-sig, df)                                 #alpha to the right#
}

pvalue_chi=function(val, dof) {                     #Probability as extreme (large) under null#
  1-pchisq(val, dof)                                #"As large" implies right tail#
}


###2a. Executing Pearson Hypothesis Test###
significance=0.05                                   #Someone somewhere liked 5%#
dof=1                                               #degrees of freedom#

myrr=rr_chi(significance, dof)                      #Reject for values greater than this (3.84)#
myteststat=teststat(a,b)                            #Note order doesn't matter. Here (11.36)#
mypvalue=pvalue_chi(myteststat, dof)                #well under 1%#
```

Figure 3.1: R Code For Hypothesis Test

```
> ###2c. McNemar's Package###
> mymatrix
     [,1] [,2]
[1,]   85   15
[2,]   40  110
> mcnemar.test(mymatrix, correct=FALSE)

        McNemar's Chi-squared test

data:  mymatrix
McNemar's chi-squared = 11.364, df = 1, p-value = 0.000749
```

Figure 3.2: R Package For McNemar's Test To Verify Above

4 Simulation

With the help of some online resources [2], we were able to simulate data to examine the power of our test under different circumstances. The simulation was conducted by trying different combinations of sample size, probability of reliefs, and correlations. Under every such combination, the draw.correlated.binary() function from R's "stats" package was used to create 1000 different observations from the data. The proportion of these 1000 simulated observations that had a test-stat (Proposition 2.0.6, Page 7) lying inside our rejection region (determined by a 5% significance level) were then recorded.

Before examining the results, it is worthwhile to show the specifics of how our simulation was conducted. The draw.correlated.binary() function works by generating (pseudo) random binary data under certain specifications. In the below code, we can see the outcomes for a sample size of 10, where the true probability of relief from drug A was 0.4, the true probability of relief from drug B was 0.6, and a correlation was chosen to be 0.1. The rows with two "1"'s correspond to the case where patients experience relief after taking both drugs A and B, the rows with "1 0" correspond to a patient who only experienced relief with drug A, etc.

```
> corrmatrix=matrix(c(1, 0.1, 0.1, 1),                    #2 by 2 matrix of correlations#
+                   byrow=TRUE, nrow=2, ncol=2)
> bindata=as.data.frame(                                  #Usually prefer working with df's#
+   draw.correlated.binary(10, d=2,                       #PRG from MultiRNG package#
+                          prop.vec=c(0.4, 0.6),          #True proportions#
+                          corr.mat=corrmatrix))          #Correlation matrix above#
> bindata
   V1 V2
1   1  1
2   0  1
3   0  0
4   0  1
5   0  1
6   1  0
7   0  1
8   0  0
9   1  0
10  1  1
```

The above would be one observation of the 1000 carried out for that specific combination of values. In all, we tested sample sizes ranging from 25 to 200, true probability differences ranging from 0 to 0.1, and correlations ranging from 0 to 0.5. The specifics are shown below, along with a user-defined function which takes as input the observation and our significance level, and outputs either a "1" (reject null hypothesis) or "0".

```
###3a. Initial specifications###
n=c(25, 40, 80, 200)                                      #Different values of n#
p1=c(0.1, 0.4, 0.8)                                       #Different mean success rates of drug A#
pertubation=c(0, 0.02, 0.05, 0.1)                         #Difference in drug efficacy#
rho=c(0, 0.2, 0.5)                                        #Different correlation values#
size=1000                                                 #How many iterations we'll test#
alpha=0.05                                                #Someone somewhere liked 5% and now we're stuck#

mytest=function(df, sig) {                               #Arguments are 2x2 df and significance level#
  reduced=df[which(df[,1] != df[,2]),]                    #Only care about discordant pairs#
  a=sum(reduced[,1]); b=sum(reduced[,2])                  #Enumerate cases#
  if(nrow(reduced)==0) {                                  #Dealing with NAs (no discordant pairs)#
    teststat=0
  } else {
    teststat=(a-b)^2/(a+b)                                #Our observed test stat#
  }
  threshold=qchisq(1-sig, 1)                              #sig% of data lies to right under null#
  ifelse(teststat>threshold, 1, 0)                        #Returns 1 if reject null#
}
```

Our simulation was conducted in a way intended to make R aficionado's skin crawl— lots of "for" loops! The below figure shows the script used to generate the $1000 \times 4 \times 3 \times 4 \times 3 = 144000$ different simulated observations.

```
###3b. Perform Simulation###
set.seed(502)                                          #To make reproducible#
count=vector(); reject=vector(); z=0                   #Initialize#
correlations=vector(); proba=vector();                 #More initializing#
samplesize=vector(); probb=vector()                    #More initializing#

for (i in 1:length(n)) {                               #For every sample size...#
  for (j in 1:length(p1)) {                            #...And every success rate of A...#
    for (k in 1:length(pertubation)) {                 #...And every success rate of B...#
      for (l in 1:length(rho)) {                       #...And every correlation...#
        for (m in 1:size) {                            #...Do a lot of iterations#
          corrmatrix=matrix(c(1, rho[l], rho[l], 1),   #2 by 2 matrix of correlations#
                            byrow=TRUE, nrow=2, ncol=2)

          bindata=as.data.frame(                       #Usually prefer working with df's#
            draw.correlated.binary(n[i], d=2,          #PRG from MultiRNG package#
                    prop.vec=c(p1[j], p1[j]+pertubation[k]),  #True proportions#
                    corr.mat=corrmatrix))              #Correlation matrix above#

          count[m]=mytest(bindata, alpha)              #Keep track of reject/fail to reject#

        }
        z=z+1
        correlations[z]=rho[l]                         #which correlation produced results#
        probb[z]=pertubation[k]+p1[j]                  #which prob B produced results
        proba[z]=p1[j]                                 #which prob A produced results#
        samplesize[z]=n[i]                             #which sample size produced results#
        reject[z]=sum(count)/size                      #% of specific n,p1, rho combos that reject#
      }
    }
  }
}

fulldf=data.frame(samplesize, proba, probb, correlations, reject)
```

First, we studied how well our tests controlled for Type I error. We artificially set our significance level at 5%, so would suspect/hope that in cases where the null hypothesis is true (i.e. the true probability of relief from A is the same as the true probability of relief from B) we would only incorrectly reject the null hypothesis 5% of the time over the long-run. The below script shows that with 1000 observations, this is largely seen to be true. In the 36 combinations of sample sizes, true probabilities, and correlations that we tried, the largest our Type I Error got was 6.3%, while the average Type I Error was below 5%.

```
> ###3c. Alpha Analysis###
> alphadf=fulldf[which(fulldf$proba == fulldf$probb),]          #Filter to cases of null hypothesis#
> alphadf
    samplesize proba probb correlations reject
1           25   0.1   0.1          0.0  0.051
2           25   0.1   0.1          0.2  0.050
3           25   0.1   0.1          0.5  0.016
13          25   0.4   0.4          0.0  0.042
14          25   0.4   0.4          0.2  0.055
15          25   0.4   0.4          0.5  0.052
25          25   0.8   0.8          0.0  0.048
26          25   0.8   0.8          0.2  0.052
27          25   0.8   0.8          0.5  0.050
37          40   0.1   0.1          0.0  0.063
38          40   0.1   0.1          0.2  0.038
39          40   0.1   0.1          0.5  0.040
49          40   0.4   0.4          0.0  0.051
50          40   0.4   0.4          0.2  0.047
51          40   0.4   0.4          0.5  0.049
61          40   0.8   0.8          0.0  0.048
62          40   0.8   0.8          0.2  0.045
63          40   0.8   0.8          0.5  0.051
73          80   0.1   0.1          0.0  0.053
74          80   0.1   0.1          0.2  0.049
75          80   0.1   0.1          0.5  0.063
85          80   0.4   0.4          0.0  0.060
86          80   0.4   0.4          0.2  0.051
87          80   0.4   0.4          0.5  0.055
97          80   0.8   0.8          0.0  0.062
98          80   0.8   0.8          0.2  0.052
99          80   0.8   0.8          0.5  0.040
109        200   0.1   0.1          0.0  0.050
110        200   0.1   0.1          0.2  0.049
111        200   0.1   0.1          0.5  0.054
121        200   0.4   0.4          0.0  0.050
122        200   0.4   0.4          0.2  0.051
123        200   0.4   0.4          0.5  0.047
133        200   0.8   0.8          0.0  0.051
134        200   0.8   0.8          0.2  0.052
135        200   0.8   0.8          0.5  0.041
> summary(alphadf$reject)                                      #Worst combo is 6.3% false positive#
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01600 0.04775 0.05050 0.04939 0.05200 0.06300
```

Figure 4.1: Alpha Control Of Simulation

Next, we studied how powerful our tests were. To do so, we stored segments of our simulation with the same sample size and true probability together as elements in a list. With this list in hand, we were able to build line plots with R's "ggplot" package, grouping together different correlations on the same plot, and adding features as desired. The below code makes clear how we did so.

```
###3d. Power Analysis###
powerdf=fulldf[which(fulldf$proba != fulldf$probb),]          #Filter to cases of alternative hypothesis#
power=rep(0.8, nrow(powerdf))                                 #80% true positive rate "pretty good"#
diff=powerdf$probb-powerdf$proba                              #For x-axis#
powerdf=data.frame(powerdf, power, diff)                      #New df for alternative hypothesis#
mylist=list(); k=0                                            #Initialize#

for (i in 1:length(p1)) {
  for (j in 1:length(n)) {
    k=k+1
    temp=powerdf[which(powerdf$samplesize==n[j] &             #Temporary df w/ specific size/prob combo#
                       powerdf$proba==p1[i]),]
    mylist[[k]]=temp[order(temp$correlations),]               #Store as element in a list#
  }
}



###3e. Visuals###
plots=list()                                                  #Initialize#

for (i in 1:length(mylist)) {                                 #Create one plot for every size/prob combo#
plots[[i]]=
  ggplot(mylist[[i]], aes(x = diff, y = reject,               #Target is rejection rate...#
  group=factor(correlations), color=factor(correlations))) + #...with different correlations in same plot#
  geom_line(linewidth=1, linetype=1) +                        #regular lines#
  geom_hline(yintercept=0.8,                                  #Create reference point of "pretty good" rate...#
             linewidth=1.25, linetype=2, color = "orange") + #...make it dashed and wider to stick out#
  scale_color_manual(                                         #Set different colors for different correlations#
    values = c("0"="red", "0.2"="blue", "0.5"="green")) +
  ylim(0,1) +                                                 #Force y-axis to span [0,1]#
  ggtitle(bquote("Power Plot (n=" *                           #Title plots to give size/prob combo#
                   .(mylist[[i]]$samplesize[1]) * ", P1=" *
                   .(mylist[[i]]$proba[1]) *")")) +
  xlab("Prob B - Prob A") +                                   #Give x-axis a title#
  ylab("Rejection Rate") +                                    #Give y-axis a title#
  theme_bw() +                                                #I abhor ggplots defaults#
  theme(legend.position = "none") +                           #See above... get rid of ugly legend#
  theme(plot.title = element_text(hjust = 0.5)) +             #Center main title#
  theme(plot.title = element_text(size = 10)) +               #Here and below make font size "fit" w/ 4 plots#
  theme(axis.text.y = element_text(size = 10))+
  theme(axis.text.x = element_text(size = 10))
}

grid.arrange(plots[[1]], plots[[2]],                          #Four plots in 1; ggplot can't do mfrow#
             plots[[3]], plots[[4]], ncol = 4)
grid.arrange(plots[[5]], plots[[6]],
             plots[[7]], plots[[8]], ncol = 4)
grid.arrange(plots[[9]], plots[[10]],
             plots[[11]], plots[[12]], ncol = 4)
```

The power plots of these results (see Figure 4.2 below) tell a few simple tales. First, a test's power is a function of the alternative— when the true difference in the effectiveness between drugs A and B is small, the test will reject less often than when the true difference is large. Second, sample size is integral to the power of a test. Note that even when the effectiveness of drug A and B truly differed by 10%, no combination of probability and correlation was sufficient to breach a power of 80% with a sample size under 200. Finally, and somewhat of a technical point, the correlation of the true probabilities when generating the simulated data was important. As correlation between variables increases, one should expect the rejection rates to increases as well. This is actually fairly important, as the whole reason why we used McNemar's test in the first place was that the data was not independent.

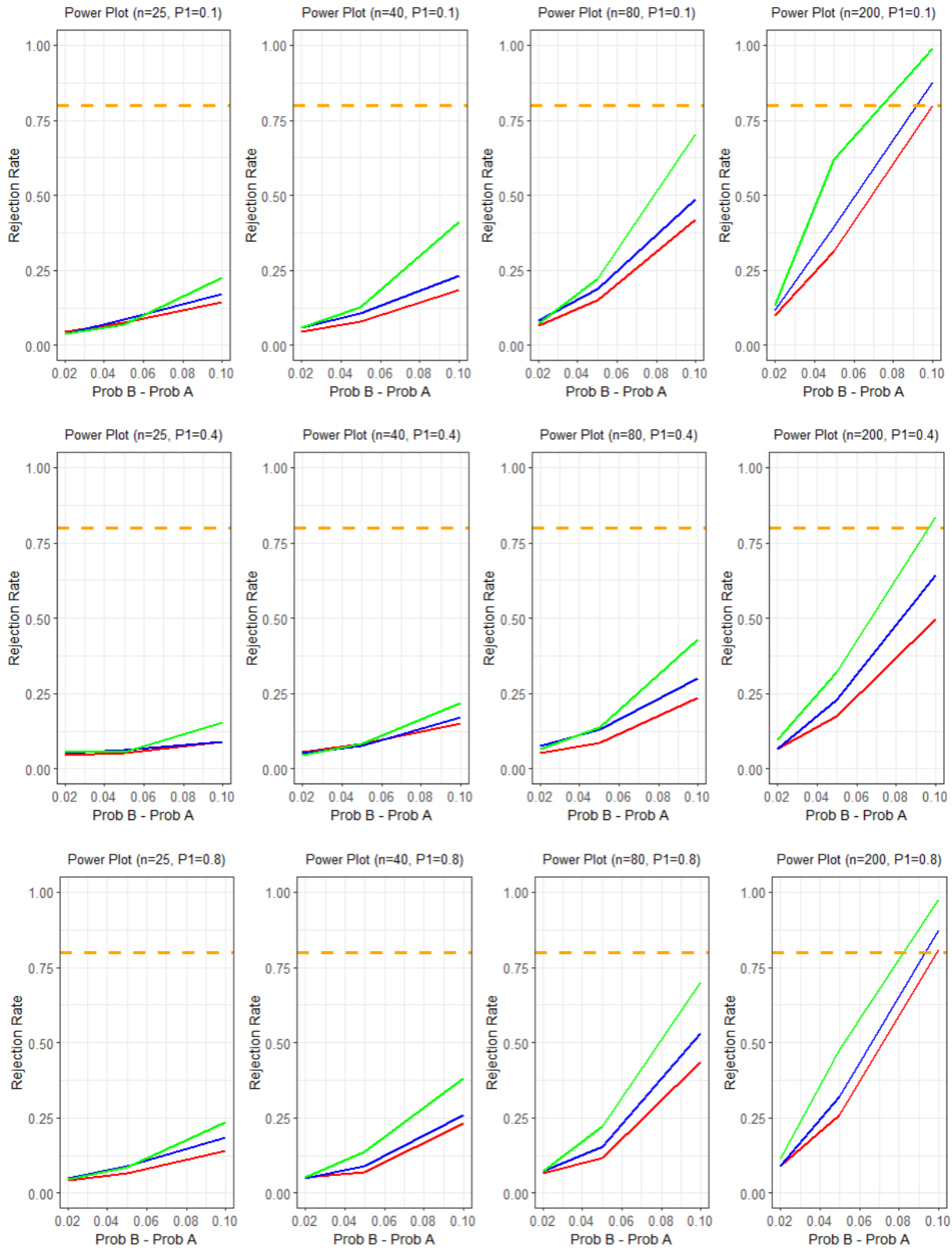

Figure 4.2: Power Plots For Different Combinations Of Size And Probabilities
Red=0.0 Correlation, Blue=0.2 Correlation, Green=0.5 Correlation

5 Full R Code

A .R file is also attached.

```
##########ST502 Project 1: Liam Flaherty##########
##########Goal is to test probability of relief from drugs A and B##########
##########Should take <90s when running everything from the top##########

#####1. Load Required Packages And Data#####
install.packages("stats")
install.packages("MultiRNG")
install.packages("ggplot2")
install.packages("gridExtra")
library(stats)
library(MultiRNG)
library(ggplot2)
library(gridExtra)

both=85; neither=110; a=40; b=15                    #e.g. 15 got better with B but not A#
mydata=c(both, b, a, neither)                       #Put data into a vector#
mydf=data.frame(c(both, a), c(b, neither))          #Put data into data frame#
mymatrix=matrix(mydata, nrow=2, ncol=2, byrow=TRUE) #Reluctantly put data into a matrix#



#####2. Hypothesis Test Set-up#####
teststat=function(discordant1, discordant2) {
  ((discordant1-discordant2)^2/(discordant1+discordant2))     #Pearson's Chi-Square#
}

rr_chi=function(sig, dof) {                         #Reject for large#
  qchisq(1-sig, dof)                                #alpha to the right#
}

pvalue_chi=function(val, dof) {                     #Probability as extreme (large) under null#
  1-pchisq(val, dof)                                #"As large" implies right tail#
}


###2a. Executing Pearson Hypothesis Test###
significance=0.05                                   #Someone somewhere liked 5%#
dof=1                                               #degrees of freedom#

myrr=rr_chi(significance, dof)                      #Reject for values greater than this (3.84)#
myteststat=teststat(a,b)                            #Note order doesn't matter. Here (11.36)#
mypvalue=pvalue_chi(myteststat, dof)                #Well under 1%#


###2b. McNemar's Package###
mymatrix
mcnemar.test(mymatrix, correct=FALSE)               #From "stats" package#



#####3. Simulation Study#####
###Load MultiRNG package###

###3a. Initial specifications###
n=c(25, 40, 80, 200)                                #Different values of n#
p1=c(0.1, 0.4, 0.8)                                 #Different mean success rates of drug A#
pertubation=c(0, 0.02, 0.05, 0.1)                   #Difference in drug efficacy#
rho=c(0, 0.2, 0.5)                                  #Different correlation values#
size=1000                                           #How many iterations we'll test#
alpha=0.05                                          #Someone somewhere liked 5% and now we're stuck#

mytest=function(df, sig) {                          #Arguments are 2x2 df and significance level#
  reduced=df[which(df[,1] != df[,2]),]              #Only care about discordant pairs#
  a=sum(reduced[,1]); b=sum(reduced[,2])            #Enumerate cases#
  if(nrow(reduced)==0) {                            #Dealing with NAs (no discordant pairs)#
    teststat=0
  } else {
    teststat=(a-b)^2/(a+b)                          #Our observed test stat#
  }
  threshold=qchisq(1-sig, 1)                        #sig% of data lies to right under null#
  ifelse(teststat>threshold, 1, 0)                  #Returns 1 if reject null#
}
```

```
###3b. Perform Simulation###
set.seed(502)                                          #To make reproducible#
count=vector(); reject=vector(); z=0                   #Initialize#
correlations=vector(); proba=vector();                 #More initializing#
samplesize=vector(); probb=vector()                    #More initializing#

for (i in 1:length(n)) {                               #For every sample size...#
  for (j in 1:length(p1)) {                            #...And every success rate of A...#
    for (k in 1:length(pertubation)) {                 #...And every success rate of B...#
      for (l in 1:length(rho)) {                       #...And every correlation...#
        for (m in 1:size) {                            #...Do a lot of iterations#
          corrmatrix=matrix(c(1, rho[l], rho[l], 1),   #2 by 2 matrix of correlations#
                            byrow=TRUE, nrow=2, ncol=2)

          bindata=as.data.frame(                       #Usually prefer working with df's#
            draw.correlated.binary(n[i], d=2,          #PRG from MultiRNG package#
                    prop.vec=c(p1[j], p1[j]+pertubation[k]),  #True proportions#
                    corr.mat=corrmatrix))              #Correlation matrix above#

          count[m]=mytest(bindata, alpha)              #Keep track of reject/fail to reject#

        }
        z=z+1
        correlations[z]=rho[l]                         #Which correlation produced results#
        probb[z]=pertubation[k]+p1[j]                  #Which prob B produced results
        proba[z]=p1[j]                                 #Which prob A produced results#
        samplesize[z]=n[i]                             #Which sample size produced results#
        reject[z]=sum(count)/size                      #% of specific n,p1, rho combos that reject#
      }
    }
  }
}

fulldf=data.frame(samplesize, proba, probb, correlations, reject)


###3c. Alpha Analysis###
alphadf=fulldf[which(fulldf$proba == fulldf$probb),]          #Filter to cases of null hypothesis#
alphadf
summary(alphadf$reject)                                       #Worst combo is 6.3% false positive#



###3d. Power Analysis###
powerdf=fulldf[which(fulldf$proba != fulldf$probb),]          #Filter to cases of alternative hypothesis#
power=rep(0.8, nrow(powerdf))                                 #80% true positive rate "pretty good"#
diff=powerdf$probb-powerdf$proba                              #For x-axis#
powerdf=data.frame(powerdf, power, diff)                      #New df for alternative hypothesis#
mylist=list(); k=0                                            #Initialize#

for (i in 1:length(p1)) {
  for (j in 1:length(n)) {
    k=k+1
    temp=powerdf[which(powerdf$samplesize==n[j] &             #Temporary df w/ specific size/prob combo#
                       powerdf$proba==p1[i]),]
    mylist[[k]]=temp[order(temp$correlations),]                #Store as element in a list#
  }
}
```

```
###3e. Visuals###
plots=list()                                                    #Initialize#

for (i in 1:length(mylist)) {                                   #Create one plot for every size/prob combo#
plots[[i]]=
  ggplot(mylist[[i]], aes(x = diff, y = reject,                 #Target is rejection rate...#
  group=factor(correlations), color=factor(correlations))) +     #...with different correlations in same plot#
  geom_line(linewidth=1, linetype=1) +                          #regular lines#
  geom_hline(yintercept=0.8,                                    #Create reference point of "pretty good" rate...#
             linewidth=1.25, linetype=2, color = "orange") +    #...make it dashed and wider to stick out#
  scale_color_manual(                                           #Set different colors for different correlations#
    values = c("0"="red", "0.2"="blue", "0.5"="green")) +
  ylim(0,1) +                                                   #Force y-axis to span [0,1]#
  ggtitle(bquote("Power Plot (n=" *                             #Title plots to give size/prob combo#
                   .(mylist[[i]]$samplesize[1]) * ", P1=" *
                   .(mylist[[i]]$proba[1]) *")")) +
  xlab("Prob B - Prob A") +                                     #Give x-axis a title#
  ylab("Rejection Rate") +                                      #Give y-axis a title#
  theme_bw() +                                                  #I abhor ggplots defaults#
  theme(legend.position = "none") +                             #See above... get rid of ugly legend#
  theme(plot.title = element_text(hjust = 0.5)) +               #Center main title#
  theme(plot.title = element_text(size = 10)) +                 #Here and below make font size "fit" w/ 4 plots#
  theme(axis.text.y = element_text(size = 10))+
  theme(axis.text.x = element_text(size = 10))
}

grid.arrange(plots[[1]], plots[[2]],                            #Four plots in 1; ggplot can't do mfrow#
             plots[[3]], plots[[4]], ncol = 4)
grid.arrange(plots[[5]], plots[[6]],
             plots[[7]], plots[[8]], ncol = 4)
grid.arrange(plots[[9]], plots[[10]],
             plots[[11]], plots[[12]], ncol = 4)
```

6 References

[1] *Mcnemar.test: Mcnemar's chi-squared test for count data RDocumentation*. Available at: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/mcnemar.test (Accessed: 20 April 2024)

[2] Howson, I. (2021) *Draw.correlated.binary: Generation of correlated binary data in multirng: Multivariate pseudo-random number generation, draw.correlated.binary: Generation of Correlated Binary Data in MultiRNG: Multivariate Pseudo-Random Number Generation*. Available at: https://rdrr.io/cran/MultiRNG/man/draw.correlated.binary.html (Accessed: 20 April 2024).