UWEO StatR301 Homework 1 Key

Assaf Oron, May 2013

Question 1

This question demonstrates via a simple example (familiar to those among you who did StatR201), the essence of numerical Null generation and the related hypothesis test. It also touches at the essence of hypothesis testing itself.

The regression Null hypothesis states

> There is no association between x and y.

(at least not of the type specified by the model)

The classic, asymptotic, frequentist test of this Null is the one appearing next to x's line item in the regression `summary`. It assumes that $\hat{\beta}$ for the x effect is asymptotically normal, with mean 0 and variance to be calculated from the data. As we've seen in StatR201 HW2, at least for linear regression with a single x, this Null is fairly robust and unbiased to non-Normality. That is, when the reality is truly Null, H_0 is rejected at the $\alpha = 0.05$level approximately 5% of the time regardless of the distribution of individual errors.

However, that might not always be the case for any model and any specific situation. Thus it is often valuable to go back to the basics and make a **randomization or label-permutation** test, or some other form of valid numerical inference.

I said, *"going back to the basics"* because we start from the Null's basic premise that x and y are not associated. If so, then scrambling one of them around and calculating the "association" is scientifically equivalent for any such scrambling. In other words, the collection of all such scramblings can help construct a **numerical Null distribution.**

Without further ado, on to the questions themselves.

1a. The actual Effect and its Asymptotic p-value

```
challenger=read.csv('../Datasets/challenger.csv',as.is=TRUE)
chal0=glm(cbind(Eroded,Intact)~tempF,data=challenger,family=binomial)
z=summary(chal0); asymp=z$coef[2,4]
print(z)
```

```
##
## Call:
## glm(formula = cbind(Eroded, Intact) ~ tempF, family = binomial,
##     data = challenger)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.753  -0.553  -0.339  -0.190   1.539
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   8.8169     3.6070    2.44    0.015 *
## tempF        -0.1795     0.0582   -3.08    0.002 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 20.706  on 22  degrees of freedom
## Residual deviance:  9.527  on 21  degrees of freedom
## AIC: 24.87
##
## Number of Fisher Scoring iterations: 6
```

Notes:

1. If you don't remember the syntax, it is unique to logistic regression. This regression (called by `glm` with `family=binomial`) can accept as its `formula` LHS:
2. a vector 0's and 1's (or TRUE/FALSE);
3. a two-column `cbind` as above, with the left column indicating the "yes" count and the right column the "no" count;
4. a vector of any values bounded in $[0,1]$ (but with a warning).
    1. The p-value in the `summary` is **two-sided.** That is, the test is essentially on $|\hat{\beta}|$ rather than on $\hat{\beta}$.

1b. The numerical Null and p-value

The most common error here was people taking an ensemble of **p-values** from the different permutations. I assume this is because the StatR201 HW I referred you to had collected p-values. However, the question here is different. We are interested in **effect size**, and whether the actual observed $\hat{\beta}$ is relatively egregious.

Therefore, we should collect the $\hat{\beta}$ values:

```
# This might speed up the run time a bit (?)
temperms=replicate(10000,sample(challenger$tempF))

chalnull=apply(temperms,2,function(x,dat=challenger) {
  dat$permT=x
  summary(glm(cbind(Eroded,Intact)~permT,data=dat,family=binomial))$coef[2,1]
})

table(abs(chalnull)>=abs(chal0$coef[2]))
```

```
##
## FALSE  TRUE
##  9931    69
```

The 2-sided numerical p-value is a bit more conservative than the asymptotic one. However, if we switch to one-sided, it becomes much smaller than half the reported asymptotic p-value:

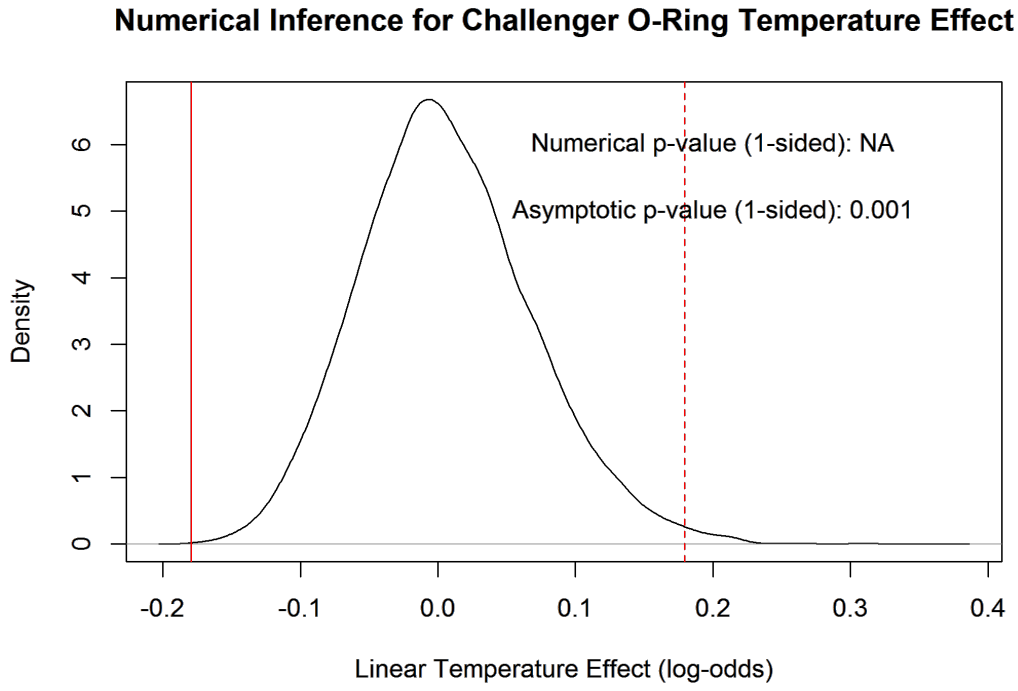
```
table(chalnull<=chal0$coef[2])
```

```
##
## FALSE
## 10000
```

Which is the right one to use? Here, a coherent argument might be made that the scientific question is one-sided. No one claimed or suspected that warm temperatures (in the range examined) might lead to o-ring failure.

Anyway, my favorite way of displaying a numerical Null and its p-value is like this:

```
plot(density(chalnull),main="Numerical Inference for Challenger O-Ring Temperature
Effect",xlab="Linear Temperature Effect (log-odds)")
abline(v=chal0$coef[2],col=2)
abline(v=-chal0$coef[2],col=2,lty=2)
text(0.2,6,paste("Numerical p-value (1-sided):",table(chalnull<=chal0$coef[2])
[2]/10000))
text(0.2,5,paste("Asymptotic p-value (1-sided):",round(asymp/2,4)))
```



Important note: this problem was made easy by being univariate, i.e. with a single x. When you have a regression with, say, two correlated covariates x_1 and x_2, you must take care to permute labels properly. It's probably better to permute entire rows of the X matrix intact (so only permute the relationship of a given set of covariates with its observation), rather than just the covariate you are interested in. This is probably what `lmPerm` does…

Question 2

Similarly to Q1, Q2 is designed to demonstrate the basic difference between Bayesian and frequentist estimation, using a very simple example.

The most common implementation mistake in the solutions, was **people making the problem more complicated than it is.** Since I chose a conjugate Beta-Binomial prior, the Bayesian posterior-mean estimate for each individual set of "coin tosses" is a simple arithmetic formula, requiring no numerical method such as integration. Specifically (as shown in class, too), the Bayesian estimate throughout Q2 is

$$\hat{p}_{Bayes} = \frac{x+3}{n+6},$$

with x being the number of "heads" out of n tosses.

Note that given each coin-toss is binary, each estimate can take on only $n+1$ distinct values. The graininess is evident with $n=5$, where the possible values for the frequentist MLE are $(0, 1/5, 2/5, 3/5, 4/5, 1)$ and for the Bayesian mean $(3/11, 4/11, 5/11, 6/11, 7/11, 8/11)$

This set of values also exposes as a **shrinkage estimator:** it is clustered around some pre-determined value. Shrinkage estimators form a broad family that is not necessarily Bayesian: for example, Lasso and ridge regression are also members (they shrink the $\hat{\beta}$'s towards zero).

Shrinkage estimators often emerge with better RMSE than the analogous pre-shrunk estimators. The benefit from variance reduction can offset even a somewhat misguided (and hence biasing) shrinkage target. They also deliver other more conceptual advantages: in the Lasso case, parsimony – and in our coin-toss case, avoidance of unrealistic estimates with small n.

Ok, to business. Here are some prep functions, including a use of our old parallel-processing friend `foreach` (sans the parallel-procesing bit):

```
rmse=function(x,ref) sqrt(mean((x-ref)^2))
bias=function(x,ref) mean(x-ref)
vari=function(x,...) var(x)  ### allows var() with ... in call
mae=function(x,ref) mean(abs(x-ref))

getMetrics=function(estimates,truth,metrix)
{
  require(foreach)
  dout=foreach(a=1:length(metrix),.combine='c') %do%
  {
    tmp=metrix[[a]](estimates,ref=truth)
    names(tmp)=names(metrix)[a]
    return(tmp)
  }
return(dout)
}
```

When you have a generic task, it's best to try and create generic and portable functions/methods for it. Using the utilities above, the function below can simulate any Beta-Binomial MLE vs. Bayes comparison in one call:

```
BetaBinomComps=function(m=1000,n,p,alph,bet,metrix=list(Bias=bias,Variance=vari,RMSE=rmse))

{
  x=rbinom(m,size=n,prob=p)
  mle=x/n
  bayes=(x+alph)/(n+alph+bet)
  lout=list()
  lout$MLE=getMetrics(estimates=mle,metrix=metrix,truth=p)
  lout$Bayes=getMetrics(estimates=bayes,metrix=metrix,truth=p)

  boxplot(mle,bayes,col=5,boxwex=0.6,main=paste("Binomial Estimates, n =",n,", p =
",p),names=c("MLE","Bayes Mean"))
  abline(h=p,col=2,lwd=2)
  return(lout)
}
```

Here are the results. If one wishes to collect the various n values into nicer tables, it can be easily achieved by assigning the calls to objects in a list, then running `sapply` over them.

Question 2: Results

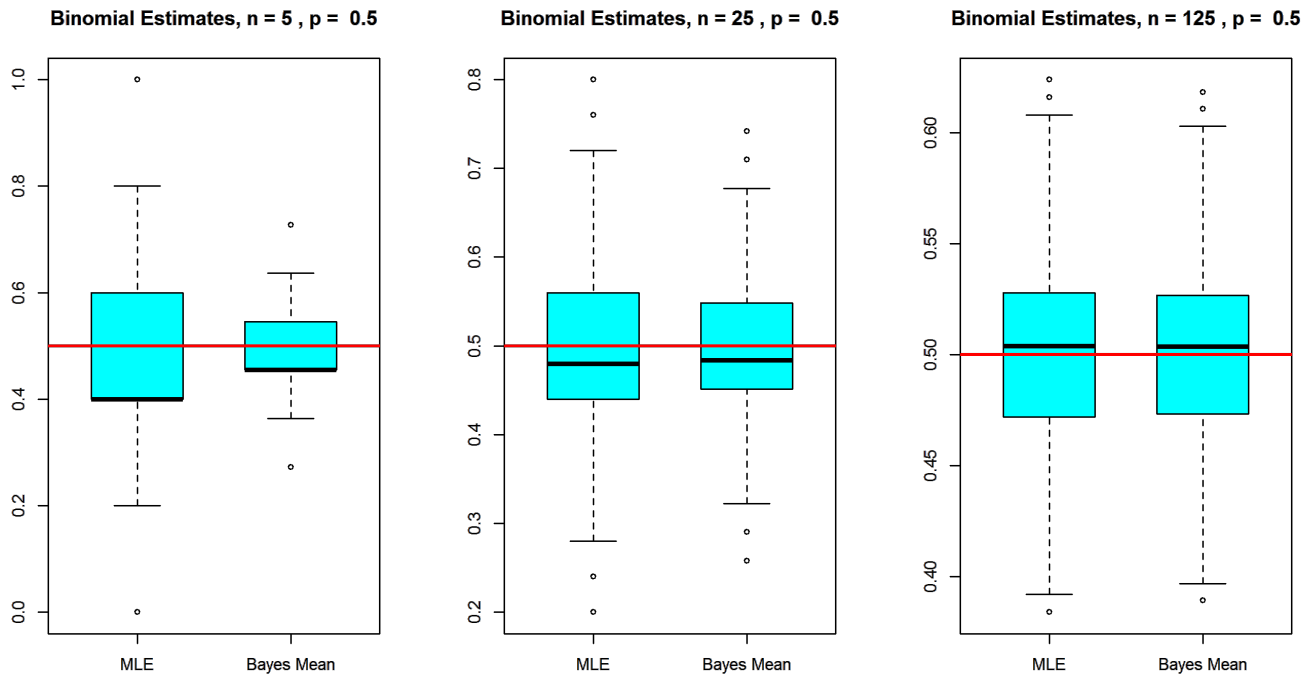
```
par(mfrow=c(1,3))
BetaBinomComps(n=5,p=.5,alph=3,bet=3)
```

```
## $MLE
##      Bias Variance      RMSE
##    0.0026   0.0514   0.2266
##
## $Bayes
##      Bias Variance      RMSE
## 0.001182 0.010621 0.103012
```

```
BetaBinomComps(n=25,p=.5,alph=3,bet=3)
```

```
## $MLE
##      Bias  Variance      RMSE
## -0.002520  0.009949  0.099728
##
## $Bayes
##      Bias  Variance      RMSE
## -0.002032  0.006471  0.080426
```

```
BetaBinomComps(n=125,p=.5,alph=3,bet=3)
```




```
## $MLE
##      Bias Variance      RMSE
## 0.000696 0.001889 0.043442
##
## $Bayes
##      Bias  Variance      RMSE
## 0.0006641 0.0017196 0.0414526
```

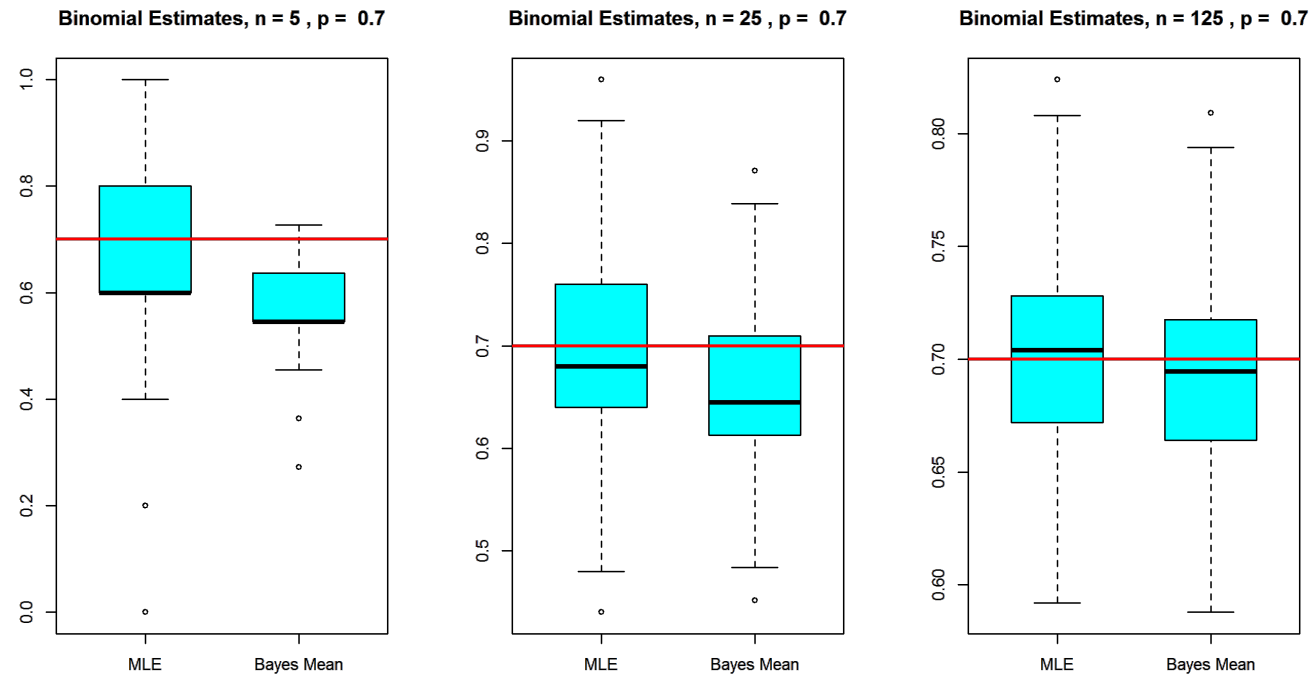
```
par(mfrow=c(1,3))
BetaBinomComps(n=5,p=.7,alph=3,bet=3)
```

```
## $MLE
##     Bias Variance     RMSE
## -0.00860  0.04029  0.20080
##
## $Bayes
##      Bias  Variance      RMSE
## -0.113000  0.008324  0.145204
```

```
BetaBinomComps(n=25,p=.7,alph=3,bet=3)
```

```
## $MLE
##      Bias  Variance      RMSE
## -0.001880  0.008145  0.090226
##
## $Bayes
##      Bias  Variance      RMSE
## -0.040226  0.005297  0.083128
```

```
BetaBinomComps(n=125,p=.7,alph=3,bet=3)
```




```
## $MLE
##     Bias Variance     RMSE
## 0.000184 0.001641 0.040487
##
## $Bayes
##      Bias  Variance      RMSE
## -0.008985  0.001494  0.039663
```

Question 2: Discussion

It is not surprising that the Bayesian estimate gets a big leg up when $p = 0.5$, perfectly matching the prior mean. Its advantage, equivalent to 6 observations, is huge with $n = 5$ and becomes almost negligible when $n = 125$. But **I was really surprised to still see the Bayesian estimate with smaller RMSEs when $p = 0.7$!**

First, this is a matter of "degree of wrongness", helped by a numerical coincidence ($n * 0.7$ is not an integer for $n = 5, 25, 125$ so $\hat{p}$ cannot match it perfectly). See this:

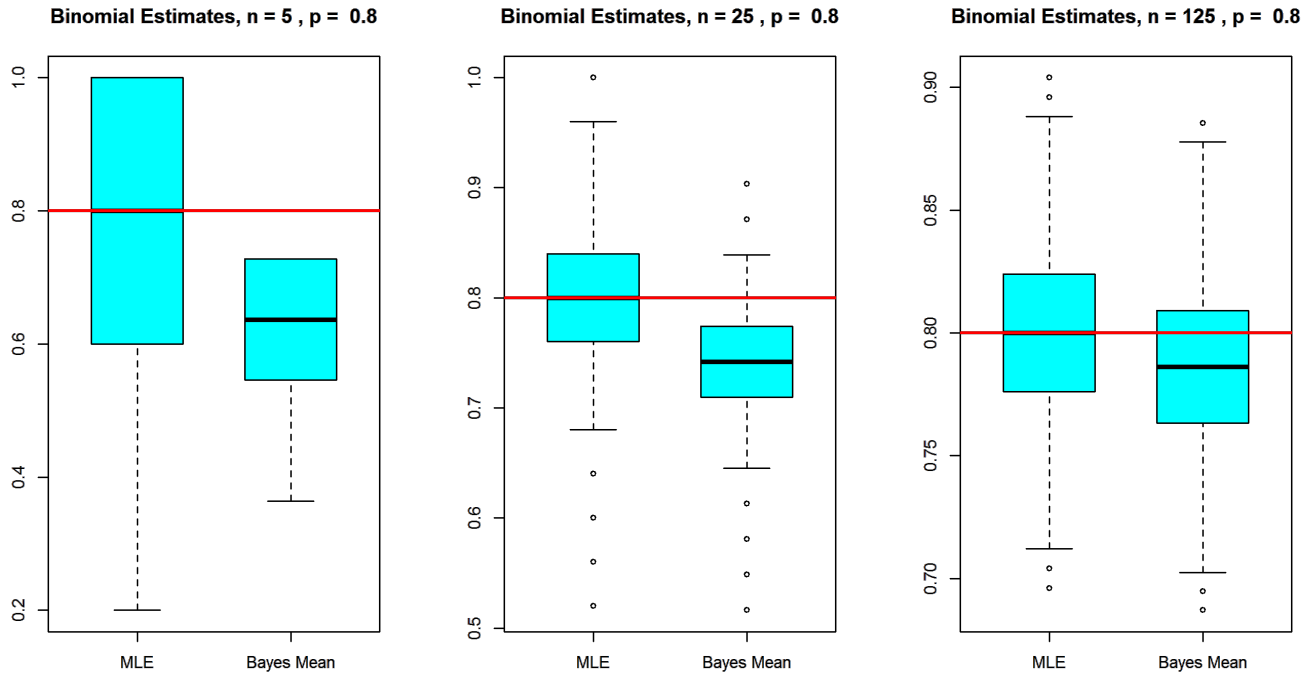
```
par(mfrow=c(1,3))
BetaBinomComps(n=5,p=.8,alph=3,bet=3)
```

```
## $MLE
##     Bias Variance     RMSE
## -0.00880  0.03532  0.18804
##
## $Bayes
##      Bias  Variance      RMSE
## -0.167636  0.007297  0.188127
```

```
BetaBinomComps(n=25,p=.8,alph=3,bet=3)
```

```
## $MLE
##      Bias  Variance      RMSE
## -0.001640  0.006165  0.078496
##
## $Bayes
##     Bias Variance     RMSE
## -0.05939  0.00401  0.08679
```

```
BetaBinomComps(n=125,p=.8,alph=3,bet=3)
```



```
## $MLE
##      Bias  Variance      RMSE
## -0.002232  0.001184  0.034462
##
## $Bayes
##      Bias  Variance      RMSE
## -0.015870  0.001078  0.036451
```

Now the MLE remains ahead of the Bayesian mean, by a little bit.

But overall, the performance of the shrunken Bayesian estimate is rather impressive. By the way, its advantage with $p = 0.7$is retained when the RMSE metric is replaced by mean absolute error, or quantile absolute error.

I learned something!

A final note: since all estimators here have a closed form and simple distributions, in principle we could have skipped the simulation altogether, and calculated the *theoretical* bias and RMSE of all estimates. But it's a good exercise nonetheless…