LZCL (2015) PNAS Example 3

Tian Zheng

February 4, 2016

This is an R Markdown document to illustrate the simulation of example 3 used in Lo et al (2015) "Why aren't significant variables automatically good predictors." appeared in PNAS.

Calculating I score

For real data, we usually use the following functions for evaluating I score on a selected variable set with or without screening. For the simulation, we used array and matrix operator to speed things up a little bit.

```
f.list.I=function(var.list, data.x, data.y){

    kk=length(var.list)

    if(kk>1){
        xx=data.x[,as.vector(var.list)]%*%as.vector((3^(0:(kk-1))))
    }
    else{
        xx=as.matrix(data.x)[,var.list]
    }
    yy=unlist(data.y)

    #print(length(xx))
    #print(length(yy))

    #dat.mat=table(xx,yy)
    xx=as.factor(xx)

    # Globle mean and variation of Y
    y.mean=mean(yy)
    y.var=var(yy)

    # Parition means of Y and counts
    mean.vec=unlist(by(yy, xx, mean))
    n.ct.vec=unlist(table(xx))

    i.score=sum((mean.vec-y.mean)^2*n.ct.vec^2)/length(yy)/y.var

    #n.d=dat.mat[,1]
    #n.u=dat.mat[,2]
    #nn.d=sum(n.d)
    #nn.u=sum(n.u)
    #i.score=nn.d*nn.u*sum((n.d/nn.d-n.u/nn.u)^2)/(nn.d+nn.u)

    return(c(var.list, i.score))
}

f.list.screen.I=function(var.list, data.x, data.y){
```

```
    kk=length(var.list)
    mk.ind=rep(1, kk)
    I.score=NA

    #print(paste("Start screening variables:", format(var.list))

    score.pre=f.list.I(var.list, data.x, data.y)[length(var.list)+1]

    if(kk>1){
        var.list.use=1:kk
        data.x.use=data.x[,var.list]
        result.v=t(combn(var.list.use, kk-1, f.list.I, simplify=T,
              data.x=data.x.use, data.y=data.y))
        print("before/after scores")
        print(c(score.pre, max(result.v[,kk])))
        if(max(result.v[,kk])> score.pre){
            mk.ind[-result.v[which.max(result.v[,kk]),1:(kk-1)]]=0
            var.list.use=1:(kk-1)
            data.x.use=data.x[,var.list[mk.ind>0]]
            out.recur=f.list.screen.I(var.list.use, data.x.use, data.y)
            mk.ind[mk.ind>0]=mk.ind[mk.ind>0]*out.recur[(kk-1)+(1:(kk-1))]
            I.score=out.recur[length(out.recur)]
        }
        else{
            I.score=score.pre
        }

    }
    else{
        I.score=score.pre
    }

    return(c(var.list, mk.ind, I.score))
}
```

Simulation setup

For this simulation, we consider `k.0=6` SNPs being jointly studied.

```
k.0=6
```

Set the seed used for this simulation.

```
seed.used=set.seed(2764)
```

For this simulation, we first assign a "baseline" odds ratio vector for the 729 6-SNP genotypes, which will be transformed up or down for different simulated scenarios. Approximately 10% of the 729 possible genotypes ($3^6 = 729$) will have an odds ratio that is different from 1 (i.e., associated with disease). The logarithm baseline odds ratio values are randomly drawn from $N(0, 0.3^2)$, which correspond to 95% of OR fall between $[0.5, 1.8]$.

To simulate different 6-SNP scenarios with different predictivity levels, we consider a range of allele frequencies and a range of odds ratio modifiers.

Disease association set-up

For this document, we run 200 simulations for each setting to speed up the documentation preparation. For our paper, we used 2000.

```
p.step=0.01 #stepsize for MAF
or.step=0.05 #stepsize for OR modifier
# three levels of sample size (case/control)
nn.seq=c(500, 1000, 1500)

# sequence of MAF of each of the 6 SNPs.
##  p.step is the step size of this sequence.
k.pp=seq(0.15, 0.4, p.step)
# sequence of OR modifier for genotypic baseline OR.
##  or.step is the step size of this sequence
or.scale=seq(1, 2, or.step)

# For each scenario, we use 2000 simulations to evaluate
## the expected value of various statistics of interest.
BB=200
```

Step 1: simulate a random vector of baseline OR.

This baseline OR is used for all 6-SNP scenarios considered.

```
   # Simulate
    gtp.or.base=exp(rnorm(3^k.0, 0, 0.3)*rbinom(3^k.0, 1, 0.1))
```

Step 2: running simulation of different 6-SNP scenarios

```
# Initialization
   gtp.or=rep(1, 3^k.0)
   popu.odds=1/20

   results.sim=array(NA, dim=c(length(nn.seq),
                               length(or.scale),
                               length(k.pp),4))
# Genotype matrix
   gtp.list=list(1:k.0)
   for(i in 1:k.0){
       gtp.list[[i]]=c(0, 1, 2)
   }
   gtp.mat=expand.grid(gtp.list)
```

Steps of simulation for each level of MAF and OR modifier: - 2a: Compute population genotype distribution given MAF. - 2b: Compute modified odds ratios $or_\gamma = or_{base}^\gamma$ (γ is the modifier) and assign large absolute log values of odds ratios to genotypes with highest population probabilties (this step is important to create predictive 6-SNP modules.) - 2c: Based on OR, we compute the conditional distribution of genotypes among cases and controls. - 2d: Based on cond. distribution for cases and controls, we compute and simulate statistics of interest under each scenario for each specified sample size.

```
for(cc in 1:length(k.pp)){

    # Step 2a
    k.p=rep(k.pp[cc], k.0)
    k.gtp.p=cbind(k.p^2, 2*k.p*(1-k.p), (1-k.p)^2)
    gtp.p=rep(1, 3^k.0)
    for(i in 1:k.0){
        gtp.p=gtp.p*k.gtp.p[i, gtp.mat[,i]+1]
    }

    for(bb in 1:length(or.scale)){
      # Step 2b
      gtp.or[order(gtp.p)]=gtp.or.base[order(abs(log(gtp.or.base)))]
      gtp.or=exp(log(gtp.or)*or.scale[bb])

      # Step 2c
      gtp.odds=gtp.or*popu.odds
      gtp.d=gtp.odds/(1+gtp.odds)*gtp.p
      gtp.d=gtp.d/sum(gtp.d)
      gtp.u=1/(1+gtp.odds)*gtp.p
      gtp.u=gtp.u/sum(gtp.u)

      # Step 2d
      cls.rate=0.5*sum(pmax(gtp.d, gtp.u))

      for(dd in 1:length(nn.seq)){
        nn=nn.seq[dd]

        #print(c(bb, cc, dd))

        ## Data
        n.d=rmultinom(BB, nn, gtp.d)
        n.u=rmultinom(BB, nn, gtp.u)

        # Chi-square
        stat.chisq=colSums(2*(n.d-n.u)^2/(n.d+n.u+1e-10))

        # I score
        stat.I=nn*colSums((n.d/nn-n.u/nn)^2)/2

        # Training set rate
        stat.cls=colSums(pmax(n.d, n.u))/nn/2

        results.sim[dd, bb, cc,]=c(cls.rate, mean(stat.cls), #1-2
                                   mean(stat.I), #3
                                   mean(stat.chisq) #4
                                   )
    }
  }
}
```

Visualize simulation results

```
library(arrayhelpers)
```

```
## Package arrayhelpers, version 0.76-20120816
##
## If you use this package please cite it appropriately.
##    citation("arrayhelpers")
## will give you the correct reference.
##
## The project homepage is http://arrayhelpers.r-forge.r-project.org/
```

```
library(lattice)
```

```
#pdf(file="out1rate.pdf", width=9, height=2.5)
#par(mfrow=c(4,3), cex.main=1, font.main=1, mar=c(1,1,1,1))
```

Convert array to matrix for plotting.

```
## True Bayes Rate
results.flat.1=array2df(results.sim[,,,1],
                        levels=list(sample.size=format(nn.seq),
                                    or.scale=or.scale,
                                    MAF=k.pp),
                        label.x="Bayes.rate")

results.flat.1[,2]=factor(x=as.character(results.flat.1[,2]),
                            levels=format(nn.seq),
                            labels=paste(nn.seq, "cases and", nn.seq, "controls"))
results.flat.1[,3]=as.numeric(as.character(results.flat.1[,3]))
results.flat.1[,4]=as.numeric(as.character(results.flat.1[,4]))

### PR's I###
results.flat.3=array2df(results.sim[,,,3],
                        levels=list(sample.size=format(nn.seq),
                                    or.scale=or.scale,
                                    MAF=k.pp), label.x="PR.I")
results.flat.3[,2]=factor(x=as.character(results.flat.3[,2]),
                            levels=format(nn.seq),
                          labels=paste(nn.seq, "cases and", nn.seq, "controls"))
results.flat.3[,3]=as.numeric(as.character(results.flat.3[,3]))
results.flat.3[,4]=as.numeric(as.character(results.flat.3[,4]))

### Training rate ####

results.flat.2=array2df(results.sim[,,,2],
                        levels=list(sample.size=format(nn.seq),
                                    or.scale=or.scale,
                                    MAF=k.pp),
                        label.x="Training.rate")
```

```
results.flat.2[,2]=factor(x=as.character(results.flat.2[,2]),
                          levels=format(nn.seq),
                         labels=paste(nn.seq, "cases and", nn.seq, "controls"))
results.flat.2[,3]=as.numeric(as.character(results.flat.2[,3]))
results.flat.2[,4]=as.numeric(as.character(results.flat.2[,4]))
```

Making the plots

```
library(lattice)
library(latticeExtra)
```

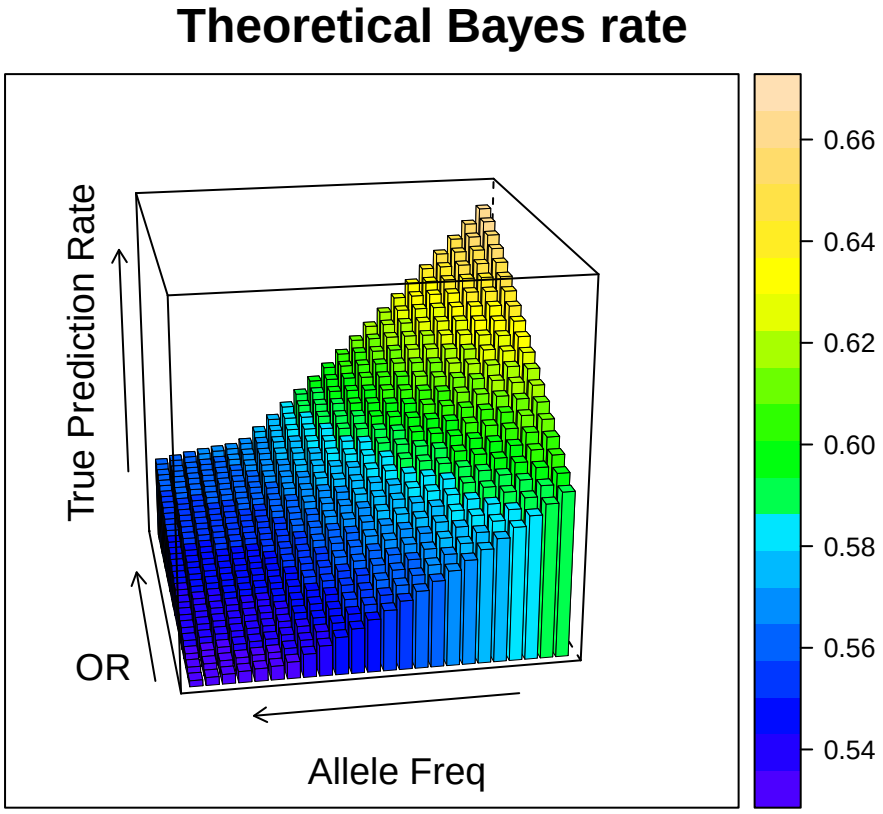
```
## Loading required package: RColorBrewer
```

```
#png("bayesrate.png", width=400, height=400)
mypanel <- function(x,y,z,...){

    panel.3dbars(x,y,z, ...,
    col.facet=panel.col[trellis.panelArgs()$subscripts])

}
view.ang=list(z=100, x=-70)

n.cut=20
panel.col=level.colors(results.flat.1$Bayes.rate[results.flat.1$sample.size==levels(results.flat.1$samp
        at = do.breaks(range(results.flat.1$Bayes.rate), n.cut),
                            col.regions = topo.colors,
                            colors = TRUE)
cloud(Bayes.rate ~ or.scale * MAF, data=results.flat.1[results.flat.1$sample.size==levels(results.flat.
     xlab=list(label="OR", cex=1.2),
     ylab=list(label="Allele Freq", cex=1.2),
     zlab=list(label="True Prediction Rate", cex=1.2, rot=90), cex.axis=0.7,
     panel.3d.cloud=mypanel, xbase=or.step*0.8, ybase=p.step*0.8,
      main=list(label="Theoretical Bayes rate", cex=1.5),
     screen=view.ang, #layout=c(3,1),
     colorkey = list(col = topo.colors,
                     at = do.breaks(range(results.flat.1$Bayes.rate), n.cut)), lwd=0.4)
```

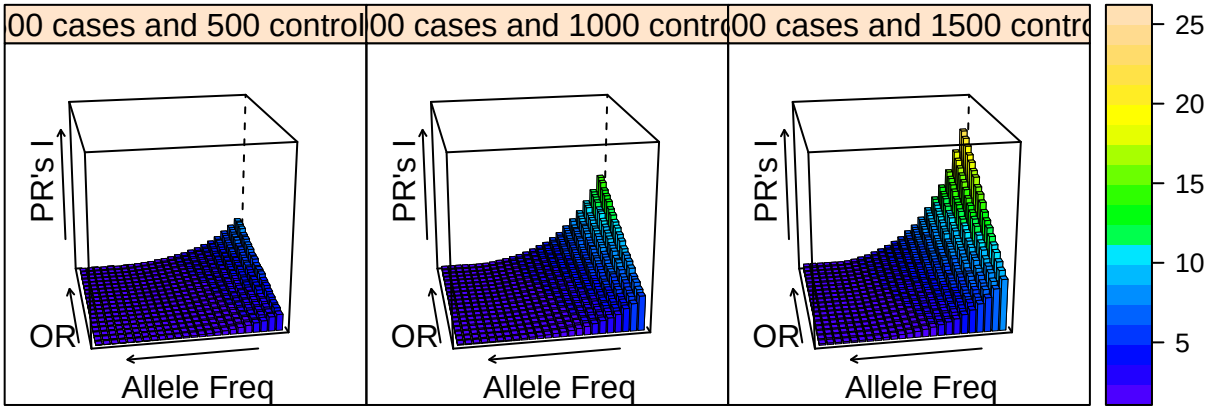



```
#dev.off()

#png("PRI.png", width=1200, height=400)
n.cut=20
panel.col=level.colors(results.flat.3$PR.I,
        at = do.breaks(range(results.flat.3$PR.I), n.cut),
                                col.regions = topo.colors,
                                colors = TRUE)

cloud(PR.I ~ or.scale * MAF|sample.size, data=results.flat.3,
     xlab=list(label="OR", cex=1),
     ylab=list(label="Allele Freq", cex=1),
     zlab=list(label="PR\'s I", cex=1, rot=90), cex.axis=0.7,
     panel.3d.cloud=mypanel, xbase=or.step*0.8, ybase=p.step*0.8,
      main=list(label="I score", cex=1.5),
     screen=view.ang, layout=c(3,1),
     colorkey = list(col = topo.colors,
                     at = do.breaks(range(results.flat.3$PR.I), n.cut)),
     lwd=0.4)
```

I score

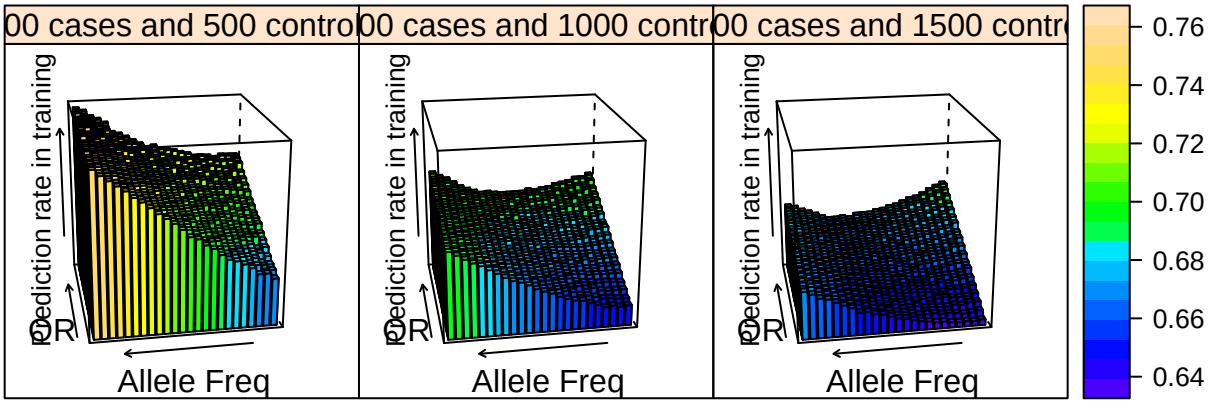



```
#dev.off()

#png("trainrate.png", width=1200, height=400)

n.cut=20
panel.col=level.colors(results.flat.2$Training.rate,
        at = do.breaks(range(results.flat.2$Training.rate), n.cut),
                                col.regions = topo.colors,
                                colors = TRUE)
cloud(Training.rate ~ or.scale * MAF|sample.size, data=results.flat.2,
     xlab=list(label="OR", cex=1),
     ylab=list(label="Allele Freq", cex=1),
     zlab=list(label="Prediction rate in training set", cex=0.8, rot=90),
     panel.3d.cloud=mypanel, xbase=or.step*0.8, ybase=p.step*0.8,
     main=list(label="In-sample training prediction rate", cex=1.5),
     screen=view.ang, layout=c(3,1),
      colorkey = list(col = topo.colors,
                      at = do.breaks(range(results.flat.2$Training.rate), n.cut)))
```

In−sample training prediction rate




```
#dev.off()
```