Computer Programs

Measuring differential treatment benefit across marker specific subgroups: the choice of outcome scale

Jaya M. Satagopan and Alexia Iasonos

Memorial Sloan Kettering Cancer Center

January 3, 2017

#############################################

#

#

# GETTING INDIVIDUAL-LEVEL DATA

#

# January 3, 2017

#

# Authors:

# Jaya M. Satagopan (satagopj@mskcc.org) and Alexia Iasonos ([iasonosa@mskcc.org](mailto:iasonosa@mskcc.org))

#

# This R function shows how to use Programs 1 and 2 to obtain

# individual-level data.

#

# Program 3, given towards the end of this document, uses the individual-level data

# to evaluate a predictive biomarker.

#

#############################################

#############

#

# First, digitize each line in Figure 1B and 1C of Larkin et al (2015, PMID: 26027431)

# to obtain the (x,y) coordinates for each line that correspond to times and

# survival probabilities.

#

# For each line, this will result in a matrix with 2 columns.

# Column 1 is time and Column 2 is survival probability for that time.

# This will be given for various time points.

#

# This will result in a total of 6 such data files, one file per line.

# There are 3 lines in Figure 1B and 3 in Figure 1C or Larkin et al. Hence, 6 files.

#

# We have named 6 these files as follows:

#

# pdl1-negative-nivo.txt, pdl1-negative-ipi.txt, pdl1-negative-combo.txt,

# pdl1-positive-nivo.txt, pdl1-positive-ipi.txt, pdl1-positive-combo.txt

#

# The R object digitized.file.names, given below, contains the names of these files.

#

##############

digitized.file.names = c("pdl1-negative-nivo.txt", "pdl1-negative-ipi.txt",

"pdl1-negative-combo.txt", "pdl1-positive-nivo.txt",

"pdl1-positive-ipi.txt", "pdl1-positive-combo.txt")

################################

#

# Now, look below Figures 1B and 1C of Larkin et al and write down the

# “number at risk” data given for various time points for each line.

#

# The list “numbers.below.figure”, shown below, contains these data.

#

################################

numbers.below.figure = list(

pdl1.negative.nivo = c(208, 192, 178, 108, 105, 98, 88, 80,

76, 74, 63, 50, 31, 24, 9, 5, 4, 2, 1, 1) ,

pdl1.negative.ipi = c(202, 183, 166, 82, 72, 59, 44, 39, 35,

31, 26, 22, 12, 8, 3, 1),

pdl1.negative.combo = c(210, 195, 181, 142, 134, 123, 112, 106,

105, 96, 88, 79, 42, 36, 13, 9, 6, 2, 1),

pdl1.positive.nivo = c(80, 76, 71, 57, 56, 54, 51, 49, 49, 43, 38,

32, 16, 13, 5, 4, 2),

pdl1.positive.ipi = c(75, 69, 66, 40, 33, 24, 22, 21, 21, 17, 16, 15,

9, 6, 3, 2, 2),

pdl1.positive.combo = c(68, 63, 61, 53, 52, 47, 44, 42, 42, 39,

34, 24, 16, 12, 3, 1, 1)

)

####################

#

# Now, specify how far along on the x-axis of each line we want to go to extract data.

# For example, in one line we may want to go up to time 15 units,

# in another line up to time 18 units etc.

#

# As above, organize these times (integer values) for each line in the same order

# as the sheets in the excel file.

#

####################

time = list( time.pdl1.neg.nivo = 0:18,

time.pdl1.neg.ipi = 0:15,

time.pdl1.neg.combo = 0:18,

time.pdl1.pos.nivo = 0:17,

time.pdl1.pos.ipi = 0:17,

time.pdl1.pos.combo = 0:17

)

############################################

#

# Now reconstruct the individual-level data using the R objects created above.

#

############################################

################

#

# arm indicator

#

# 1 = pdl1-neg-nivo

# 2 = pdl1-neg-ipi

# 3 = pdl1-neg-combo

# 4 = pdl1-pos-nivo

# 5 = pdl1-pos-ipi

# 6 = pdl1-pos-combo

#

################

##############

#

# The R functions preprocess.digitized.data (Program 1)

# and Guyot.individual.data (Program 2) are given below.

# Read them into R first. Then execute the commands below to get “individual.data”.

#

##############

individual.data = NULL

for(ifile in 1:length(digitized.file.names)){

digitized.line = read.table(digitized.file.names[ifile], header=T)

processed.line.data = preprocess.digitized.data(digitized.line,

numbers.below.figure[[ifile]],

time[[ifile]])

individual.line.data = Guyot.individual.data(processed.line.data$condensed.data.set,

processed.line.data$nrisk.data,

input.arm.id=ifile)

individual.data = rbind(individual.data, individual.line.data)

}

treatment.type = c(

rep("nivolumab", length(which(individual.data[,"tmt.arm.number"] == 1))),

rep("ipilimumab", length(which(individual.data[,"tmt.arm.number"] == 2))),

rep("combination", length(which(individual.data[,"tmt.arm.number"] == 3))),

rep("nivolumab", length(which(individual.data[,"tmt.arm.number"] == 4))),

rep("ipilimumab", length(which(individual.data[,"tmt.arm.number"] == 5))),

rep("combination", length(which(individual.data[,"tmt.arm.number"] == 6))))

pdl1.status = c(rep("negative", length(which(individual.data[,"tmt.arm.number"] < 4))),

rep("positive", length(which(individual.data[,"tmt.arm.number"] >= 4))))

individual.data = as.data.frame(individual.data)

individual.data$treatment.type = treatment.type

individual.data$pdl1.status = pdl1.status

######## THE ABOVE COMMANDS GIVE THE INDIVIDUAL LEVEL DATA #####

########

######## THESE COMMANDS USE 2 R PROGRAMS GIVEN BELOW. ########

########

##################################

#

# **PROGRAM 1: R Function preprocess.digitized.data**

#

# R FUNCTION TO PREPROCESS THE DIGITIZED DATA BEFORE IT

# CAN BE USED BY THE GUYOT R CODE (Program 2) TO GET INDIVIDUAL

# LEVEL DATA

#

#

# The input to this function are:

#

# line.data = a matrix with 2 columns containing the time axis (column 1)

# and survival probability (column 2) for the digitized line

#

# num.below.fig = the numbers below the Kaplan-Meier survival curve,

# indicating the number at risk at each time point

#

# time.points = integer values of time points of interest along the x-axis.

# For example, if the figure shows x-axis up to 10,

# set time.point = 0:10

#

# max.surv.prob = maximum value of survival probability.

# If the data are digitized to get survival probability on the 0 to 100

# scale, set max.surv.prob = 100. If the data are extracted in the 0

# to 1 scale of survival probability, set max.surv.prob = 1

#

#

##################################

preprocess.digitized.data = function(line.data, num.below.fig, time.points, max.surv.prob=100){

condensed.data.set = NULL

unique.surv.prob = unique(line.data[,2])

################

# The digitized data typically will have a large number of x-axis values with the same y-axis value.

# The following commands handle this to give a clean data set with not too many repetitions

# of y-axis with different x-axis.

################

for(i in 1:length(unique.surv.prob)){

sub.data = line.data[which(line.data[,2]==unique.surv.prob[i]),]

if(nrow(sub.data)==1){

condensed.data.set = rbind(condensed.data.set, sub.data)

}

if(nrow(sub.data) > 1){

condensed.data.set = rbind(condensed.data.set, sub.data[1,])

}

}

##############

# create appropriate condensed.data.set so that as time increases,

# survival is non-increasing

##############

orig.condensed.data.set = condensed.data.set

condensed.time = unique(orig.condensed.data.set[,1])

new.condensed.data = NULL

temp.max = max.surv.prob

for(i in 1:length(condensed.time)){

temp.time = condensed.time[i]

temp.mat = orig.condensed.data.set[which(orig.condensed.data.set[,1] == temp.time),]

if(nrow(temp.mat) > 1){

temp.id = which(temp.mat[,2] <= temp.max)

if(length(temp.id) > 0){

temp.row = temp.mat[temp.id[1],]

new.condensed.data = rbind(new.condensed.data, temp.row)

temp.max = temp.row[2]

}

}

else{

temp.row = temp.mat

if(temp.row[2] <= temp.max){

new.condensed.data = rbind(new.condensed.data, temp.row)

temp.max = temp.row[2]

}

}

}

condensed.data.set = new.condensed.data

unique.condensed.time = unique(round(condensed.data.set[,1]))

event.time = NULL

temp.time = condensed.data.set[,1]

##################

#

# extract nrisk information

#

##################

lower <- NULL

upper <- NULL

for(i in 1:(length(time.points)-1)){

my.time = time.points[i]

my.id = which(temp.time >= my.time & temp.time < time.points[(i+1)])

if(length(my.id) > 0){

event.time = c(event.time, my.time)

lower = c(lower, my.id[1])

upper = c(upper, my.id[length(my.id)])

}

}

nrisk.data = data.frame(t.risk=event.time, lower=lower, upper=upper, n.risk=num.below.fig[event.time+1])

result = list(condensed.data.set=condensed.data.set, nrisk.data=nrisk.data)

return(result)

}

######################################################################

#

# **PROGRAM 2: R Function Guyot.individual.data**

#

#

# R FUNCTION TO GET INDIVIDUAL PATIENT DATA USING

# THE PREPROCESSED DIGITIZED DATA.

#

# This R function is developed entirely by

# Guyot et al (2012), BMC Medical Research Methodology, 12: 9, PMID:22297116

#

# We have converted it to a function called “Guyot.individual.data”

# for ease of use to extract data from multiple lines, as needed in our applications.

#

#

# The input are:

# condensed.data = preprocessed digitized data

#

# nrisk.data = summary of number of patients at risk, as required for this function.  
# Details are in the Guyot et al (2012) paper.

#

# input.arm.id = treatment/biomarker arm.

#

# tot.events = total number of events, if given in the paper that reports the figure.

# “NA” if it is not reported in the paper.

#

#############

Guyot.individual.data = function(condensed.data, nrisk.data, input.arm.id, tot.events="NA"){

#Algorithm to create a raw dataset from DigizeIt readings from a Kaplan-Meier curve

library("MASS")

library("splines")

library("survival")

############################################

#Read in survival times read by digizeit

############################################

digizeit<- condensed.data

t.S<-digizeit[,1]

orig.S<-digizeit[,2]/100

S = orig.S

S[1] = orig.S[1]

for(i in 2:length(orig.S)){

if(S[i] <= S[i-1]) S[i] = S[i]

if(S[i] > S[i-1]) S[i] = S[i-1]

}

arm.id = input.arm.id

#Read in published numbers at risk, n.risk, at time, t.risk, lower and upper

# indexes for time interval

pub.risk<-nrisk.data

t.risk<-pub.risk[,1]

lower<-pub.risk[,2]

upper<-pub.risk[,3]

n.risk<-pub.risk[,4]

n.int<-length(n.risk)

n.t<- upper[n.int]

#Initialise vectors

arm<-rep(arm.id,n.risk[1])

n.censor<- rep(0,(n.int-1))

n.hat<-rep(n.risk[1]+1,n.t)

cen<-rep(0,n.t)

d<-rep(0,n.t)

KM.hat<-rep(1,n.t)

last.i<-rep(1,n.int)

sumdL<-0

if (n.int > 1){

#Time intervals 1,...,(n.int-1)

for (i in 1:(n.int-1)){

# print(paste("i is ", i))

#First approximation of no. censored on interval i

n.censor[i]<- round(n.risk[i]\*S[lower[i+1]]/S[lower[i]]- n.risk[i+1])

# print(paste("n.censor[i] is ", n.censor[i]))

# ############# ADDED BY JAYA #############

if(n.censor[i] <= 0){

n.censor[i] <- 0

cen[lower[i]:upper[i]] <- 0

}

# ############ END ADDED BY JAYA #############

#Adjust tot. no. censored until n.hat = n.risk at start of interval (i+1)

while((n.hat[lower[i+1]]>n.risk[i+1])||((n.hat[lower[i+1]]<n.risk[i+1])&&(n.censor[i]>0))){

if (n.censor[i]<=0){

cen[lower[i]:upper[i]]<-0

n.censor[i]<-0

}

if (n.censor[i]>0){

cen.t<-rep(0,n.censor[i])

for (j in 1:n.censor[i]){

cen.t[j]<- t.S[lower[i]] +

j\*(t.S[lower[(i+1)]]-t.S[lower[i]])/(n.censor[i]+1)

}

#Distribute censored observations evenly over time. Find no. censored on each time interval.

cen[lower[i]:upper[i]]<-hist(cen.t,breaks=t.S[lower[i]:lower[(i+1)]], plot=F)$counts

}

#Find no. events and no. at risk on each interval to agree with K-M estimates read from curves

n.hat[lower[i]]<-n.risk[i]

last<-last.i[i]

for (k in lower[i]:upper[i]){

if (i==1 & k==lower[i]){

d[k]<-0

KM.hat[k]<-1

}

else {

d[k]<-round(n.hat[k]\*(1-(S[k]/KM.hat[last])))

KM.hat[k]<-KM.hat[last]\*(1-(d[k]/n.hat[k]))

}

n.hat[k+1]<-n.hat[k]-d[k]-cen[k]

if (d[k] != 0) last<-k

}

n.censor[i]<- n.censor[i]+(n.hat[lower[i+1]]-n.risk[i+1])

# ############# ADDED BY JAYA #############

if(n.censor[i] <= 0){

n.censor[i] <- 0

cen[lower[i]:upper[i]] <- 0

}

# ############ END ADDED BY JAYA #############

}

if (n.hat[lower[i+1]]<n.risk[i+1]) n.risk[i+1]<-n.hat[lower[i+1]]

last.i[(i+1)]<-last

}

}

#Time interval n.int.

if (n.int>1){

#Assume same censor rate as average over previous time intervals.

n.censor[n.int]<- min(round(sum(n.censor[1:(n.int-1)])\*(t.S[upper[n.int]]-

t.S[lower[n.int]])/(t.S[upper[(n.int-1)]]-t.S[lower[1]])), n.risk[n.int])

}

if (n.int==1){n.censor[n.int]<-0}

if (n.censor[n.int] <= 0){

cen[lower[n.int]:(upper[n.int]-1)]<-0

n.censor[n.int]<-0

}

if (n.censor[n.int]>0){

cen.t<-rep(0,n.censor[n.int])

for (j in 1:n.censor[n.int]){

cen.t[j]<- t.S[lower[n.int]] +

j\*(t.S[upper[n.int]]-t.S[lower[n.int]])/(n.censor[n.int]+1)

}

cen[lower[n.int]:(upper[n.int]-1)]<-hist(cen.t,breaks=t.S[lower[n.int]:upper[n.int]],

plot=F)$counts

}

#Find no. events and no. at risk on each interval to agree with K-M estimates read from curves

n.hat[lower[n.int]]<-n.risk[n.int]

last<-last.i[n.int]

for (k in lower[n.int]:upper[n.int]){

if(KM.hat[last] !=0){

d[k]<-round(n.hat[k]\*(1-(S[k]/KM.hat[last])))

}

else {d[k]<-0}

KM.hat[k]<-KM.hat[last]\*(1-(d[k]/n.hat[k]))

n.hat[k+1]<-n.hat[k]-d[k]-cen[k]

#No. at risk cannot be negative

if (n.hat[k+1] < 0) {

n.hat[k+1]<-0

cen[k]<-n.hat[k] - d[k]

}

if (d[k] != 0) last<-k

}

#If total no. of events reported, adjust no. censored so that total no. of events agrees.

if (tot.events != "NA"){

if (n.int>1){

sumdL<-sum(d[1:upper[(n.int-1)]])

#If total no. events already too big, then set events and censoring = 0 on all further time intervals

if (sumdL >= tot.events){

d[lower[n.int]:upper[n.int]]<- rep(0,(upper[n.int]-lower[n.int]+1))

cen[lower[n.int]:(upper[n.int]-1)]<- rep(0,(upper[n.int]-lower[n.int]))

n.hat[(lower[n.int]+1):(upper[n.int]+1)]<- rep(n.risk[n.int],(upper[n.int]+1-lower[n.int]))

}

}

#Otherwise adjust no. censored to give correct total no. events

if ((sumdL < tot.events)|| (n.int==1)){

sumd<-sum(d[1:upper[n.int]])

while ((sumd > tot.events)||((sumd< tot.events)&&(n.censor[n.int]>0))){

n.censor[n.int]<- n.censor[n.int] + (sumd - tot.events)

if (n.censor[n.int]<=0){

cen[lower[n.int]:(upper[n.int]-1)]<-0

n.censor[n.int]<-0

}

if (n.censor[n.int]>0){

cen.t<-rep(0,n.censor[n.int])

for (j in 1:n.censor[n.int]){

cen.t[j]<- t.S[lower[n.int]] +

j\*(t.S[upper[n.int]]-t.S[lower[n.int]])/(n.censor[n.int]+1)

}

cen[lower[n.int]:(upper[n.int]-1)]<-hist(cen.t,breaks=t.S[lower[n.int]:upper[n.int]],

plot=F)$counts

}

n.hat[lower[n.int]]<-n.risk[n.int]

last<-last.i[n.int]

for (k in lower[n.int]:upper[n.int]){

d[k]<-round(n.hat[k]\*(1-(S[k]/KM.hat[last])))

KM.hat[k]<-KM.hat[last]\*(1-(d[k]/n.hat[k]))

if (k != upper[n.int]){

n.hat[k+1]<-n.hat[k]-d[k]-cen[k]

#No. at risk cannot be negative

if (n.hat[k+1] < 0) {

n.hat[k+1]<-0

cen[k]<-n.hat[k] - d[k]

}

}

if (d[k] != 0) last<-k

}

sumd<- sum(d[1:upper[n.int]])

}

}

}

###### Guyot's write statement commented out here inside the function since we are only

###### interested in the individual-level data, which comes below as IPD

#write.table(matrix(c(t.S,n.hat[1:n.t],d,cen),ncol=4,byrow=F),paste(path,KMdatafile,sep=""),sep="\t")

### Now form IPD ###

#Initialise vectors

t.IPD<-rep(t.S[n.t],n.risk[1])

event.IPD<-rep(0,n.risk[1])

#Write event time and event indicator (=1) for each event, as separate row in t.IPD and event.IPD

k=1

for (j in 1:n.t){

if(d[j]!=0){

t.IPD[k:(k+d[j]-1)]<- rep(t.S[j],d[j])

event.IPD[k:(k+d[j]-1)]<- rep(1,d[j])

k<-k+d[j]

}

}

#Write censor time and event indicator (=0) for each censor, as separate row in t.IPD and event.IPD

for (j in 1:(n.t-1)){

if(cen[j]!=0){

t.IPD[k:(k+cen[j]-1)]<- rep(((t.S[j]+t.S[j+1])/2),cen[j])

event.IPD[k:(k+cen[j]-1)]<- rep(0,cen[j])

k<-k+cen[j]

}

}

#Output IPD

IPD<-matrix(c(t.IPD,event.IPD,arm),ncol=3,byrow=F)

colnames(IPD) = c("time", "event", "tmt.arm.number")

###### Guyot's write statement commented out since we will return the IPD data

#write.table(IPD,paste(path,KMdataIPDfile,sep=""),sep="\t", quote=F, row.names=F)

return(IPD)

}

**Table 1. Sample digitized data for PDL negative nivolumab patients. The first 20 lines of these digitized data are shown as R output:**

> digitized.line[1:20,]

x y

1 0.000 100.0

2 0.678 97.6

3 0.693 97.6

4 0.705 97.6

5 0.718 97.6

6 0.730 97.6

7 0.743 97.6

8 0.755 97.6

9 0.768 97.6

10 0.780 97.6

11 0.793 97.6

12 0.805 97.6

13 0.818 97.6

14 0.830 97.6

15 0.843 97.6

16 0.855 97.6

17 0.870 97.6

18 0.881 97.6

19 0.897 97.5

20 0.903 97.3

**Table 2. Sample individual patient data obtained using digitized data such as those shown above and the R functions given above. Sample data are shown for the first 5 patients and the last 5 as R output:**

> individual.data[1:5,]

time event tmt.arm.number treatment.type pdl1.status

1 0.678 1 1 nivolumab negative

2 0.678 1 1 nivolumab negative

3 0.678 1 1 nivolumab negative

4 0.678 1 1 nivolumab negative

5 0.678 1 1 nivolumab negative

> individual.data[839:843,]

time event tmt.arm.number treatment.type pdl1.status

839 16.1 0 6 combination positive

840 16.1 0 6 combination positive

841 16.1 0 6 combination positive

842 16.1 0 6 combination positive

843 16.1 0 6 combination positive

################################

#

# **PROGRAM 3: R Function predictive.values**

#

# R FUNCTIONS TO ESTIMATE THE PREDICTIVE VALUES

#

# THE INPUTS FOR THIS FUNCTION ARE THE VARIOUS COLUMNS OF

# individual.data, GIVEN SEPARATELY AS TIME, EVENT ETC.

#

# THIS FUNCTION FOCUSES ON BINARY BIOMARKER AND BINARY TREATMENT.

#

# Jaya M. Satagopan and Alexia Iasonos

#

#################################

predictive.values = function(time, event, treatment, biomarker, tmt.arm, anal.time=5.34){

tmt.vals = sort(unique(treatment))

bio.vals=sort(unique(biomarker))

tmt.names = as.character(tmt.vals)

bio.names = as.character(bio.vals)

####### HTB ####

htb.fit = coxph(Surv(time, event) ~

(as.character(treatment)==tmt.names[1]) \* biomarker)

fit.coeff = summary(htb.fit)$coeff

log.htb.est = fit.coeff[3,1]

log.htb.se = fit.coeff[3,3]

test.log.htb = fit.coeff[3,4]

p.log.htb = fit.coeff[3,5]

###### RTB ####

fit <- survfit(Surv(time, event) ~ tmt.arm)

t = cbind(fit$time,fit$surv,fit$std.err,fit$n.risk,fit$n.event,fit$n.censor)

n1=cumsum(fit$strata)[1] #stratum 1:B=-, trt=N

n2=cumsum(fit$strata)[2] #stratum 2:B=-, trt=C

n3=cumsum(fit$strata)[3] #stratum 3:B=+, trt=N

n4=nrow(t)

# extract surv etsimate for stratum 1

try=t[1:n1,]

est=matrix(try[try[,1]<=t1,],ncol=6) # for time t1

nnn=c(est[nrow(est),4],sum(est[,5])-1)

st1\_00=c(est[nrow(est),1:3],nnn) # estimated KM value at time =t1 for strata 1

# which here is T==0

# extract surv etsimate for stratum 2

try=t[n1+1:(n2-n1),]

est=matrix(try[try[,1]<=t1,],ncol=6) # for time t1

nnn=c(est[nrow(est),4],sum(est[,5])-1)

st1\_01=c(est[nrow(est),1:3],nnn)

# extract surv etsimate for stratum 3

try=t[n2+1:(n3-n2),] # for time t1

est=matrix(try[try[,1]<=t1,],ncol=6)

nnn=c(est[nrow(est),4],sum(est[,5])-1)

st1\_10=c(est[nrow(est),1:3],nnn)

# extract surv etsimate for stratum 4

try=t[n3+1:(n4-n3),] # for time t1

est=matrix(try[try[,1]<=t1,],ncol=6)

nnn=c(est[nrow(est),4],sum(est[,5])-1)

st1\_11=c(est[nrow(est),1:3],nnn)

est\_surv=rbind(st1\_00,st1\_01,st1\_10,st1\_11)

#has all numbers we need, time, surv estimates and st error

######### RTB ##########

log.RTB= log( (est\_surv[4,2]\*est\_surv[1,2])/(est\_surv[2,2]\*est\_surv[3,2]) )

var.log.RTB = est\_surv[4,3]^2+est\_surv[3,3]^2+est\_surv[2,3]^2+est\_surv[1,3]^2

test.log.RTB = log.RTB / sqrt(var.log.RTB)

p.log.RTB = 2\*(1 - pnorm(abs(test.log.RTB)))

######## ATB ##########

ATB = est\_surv[4,2] - est\_surv[3,2] - est\_surv[2,2] + est\_surv[1,2]

var.ATB = (est\_surv[4,2] \* est\_surv[4,3])^2 + (est\_surv[3,2] \* est\_surv[3,3])^2 +

(est\_surv[2,2] \* est\_surv[2,3])^2 + (est\_surv[1,2] \* est\_surv[1,3])^2

test.ATB = ATB / sqrt(var.ATB)

p.ATB = 2\*(1-pnorm(abs(test.ATB)))

######## COLLECT THE RESULTS FOR OUTPUT

result = rbind(c(log.htb.est, log.htb.se, test.log.htb, p.log.htb),

c(log.RTB, sqrt(var.log.RTB), test.log.RTB, p.log.RTB),

c(ATB, sqrt(var.ATB), test.ATB, p.ATB))

rownames(result) = c("log.HTB", "log.RTB", "ATB")

colnames(result) = c("estimate", "std.err", "test", "p")

return(result)

}