Sports Analytics and Data Science

Winning the Game with Methods and Models

THOMAS W. MILLER
Faculty Director of Northwestern University's Predictive Analytics Program

Sports Analytics and Data Science

Winning the Game with Methods and Models

THOMAS W. MILLER

```
if(max(abs(min(unlist(part.worth.plotting.list),na.rm=TRUE)),
  max(unlist(part.worth.plotting.list),na.rm=TRUE)) <= 2) {
  max.is.less.than.40 <- FALSE
  max.is.less.than.20 <- FALSE
  max.is.less.than.4 <- FALSE
  max.is.less.than.10 <- FALSE
  max.is.less.than.2 <- TRUE
  max.is.less.than.1 <- FALSE
  }

if(max(abs(min(unlist(part.worth.plotting.list),na.rm=TRUE)),
  max(unlist(part.worth.plotting.list),na.rm=TRUE)) <= 1) {
  max.is.less.than.40 <- FALSE
  max.is.less.than.20 <- FALSE
  max.is.less.than.4 <- FALSE
  max.is.less.than.10 <- FALSE
  max.is.less.than.2 <- FALSE
  max.is.less.than.1 <- TRUE
  }

# sometimes we override the range for part-worth plotting
# this is not usually done... but it is an option
if (fix.max.to.4) {
  max.is.less.than.40 <- FALSE
  max.is.less.than.20 <- FALSE
  max.is.less.than.10 <- FALSE
  max.is.less.than.4 <- TRUE
  max.is.less.than.2 <- FALSE
  max.is.less.than.1 <- FALSE
  }
if (!max.is.less.than.1 & !max.is.less.than.2 & !max.is.less.than.4 &
  !max.is.less.than.10 & !max.is.less.than.20 & !max.is.less.than.40)
    stop("\n\nTERMINATED: Spine chart cannot plot |part-worth| > 40")

# determine point positions for plotting part-worths on spine chart
if (max.is.less.than.1 | max.is.less.than.2 | max.is.less.than.4 |
  max.is.less.than.10 | max.is.less.than.20 | max.is.less.than.40) {
# begin if-block plotting when all part-worths in absolute value
# are less than one of the tested range values
# part-worth positions for plotting
# end if-block plotting when all part-worths in absolute value
# are less than one of the tested range values
# offsets for plotting vary with the max.is.less.than setting
  if(max.is.less.than.1) {
    list.scaling <- function(x) {0.75 + x/5}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }
  if(max.is.less.than.2) {
    list.scaling <- function(x) {0.75 + x/10}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }
```

```
  if(max.is.less.than.4) {
    list.scaling <- function(x) {0.75 + x/20}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }

   if(max.is.less.than.10) {
    list.scaling <- function(x) {0.75 + x/50}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }

  if(max.is.less.than.20) {
    list.scaling <- function(x) {0.75 + x/100}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }

  if(max.is.less.than.40) {
    list.scaling <- function(x) {0.75 + x/200}
    part.worth.point.position <-
      lapply(part.worth.plotting.list,list.scaling)
    }

  part.worth.point.position <- lapply(part.worth.plotting.list,list.scaling)
  }

if (plot.framing.box) plot(c(0,0,1,1),c(0,1,0,1),xlab="",ylab="",
  type="n",xaxt="n",yaxt="n")

if (!plot.framing.box) plot(c(0,0,1,1),c(0,1,0,1),xlab="",ylab="",
  type="n",xaxt="n",yaxt="n", bty="n")

if (put.title.on.spine.chart) {
  text(c(0.50),c(0.975),pos=3,labels=title.on.spine.chart,cex=01.5)
  y.location <- 0.925  # starting position with title
  }

if (!put.title.on.spine.chart) y.location <- 0.975  # no-title start

# store top of vertical line for later plotting needs
y.top.of.vertical.line <- y.location

x.center.position <- 0.75  # horizontal position of spine

# begin primary plotting loop
# think of a plot as a collection of text and symbols on screen or paper
# we are going to construct a plot one text string and symbol at a time
# (note that we may have to repeat this process at the end of the program)
for(k in seq(along=effect.names)) {
  y.location <- y.location - large.space
```

```
  text(c(0.4),c(y.location),pos=2,
    labels=paste(effect.name.map(effect.names[k])," ",sep=""),cex=01.0)
  text(c(0.525),c(y.location),pos=2,col=color.for.printing.importance.text,
  labels=paste(" ",left.side.symbol.to.print.around.importance,
  pretty.print(
    unlist(conjoint.results$attribute.importance[effect.names[k]])),"%",
    right.side.symbol.to.print.around.importance,sep=""),cex=01.0)

# begin loop for printing part-worths
  for(m in seq(1:number.of.levels.of.attribute[k])) {
    y.location <- y.location - medium.space
    text(c(0.4),c(y.location),pos=2,
    conjoint.results$xlevel[[effect.names[k]]][m],cex=01.0)
 #   part.worth.label.data.frame[k,m],cex=01.0)

    text(c(0.525),c(y.location),pos=2,
    col=color.for.printing.part.worth.text,
    labels=paste(" ",left.side.symbol.to.print.around.part.worths,
    pretty.print(part.worth.plotting.list[[effect.names[k]]][m]),
    right.side.symbol.to.print.around.part.worths,sep=""),cex=01.0)

    points(part.worth.point.position[[effect.names[k]]][m],y.location,
      type = "p", pch = 20, col = color.for.part.worth.point, cex = 2)
    segments(x.center.position, y.location,
    part.worth.point.position[[effect.names[k]]][m], y.location,
       col = color.for.part.worth.line, lty = 1, lwd = 2)
    }
  }

y.location <- y.location - medium.space

# begin center axis and bottom plotting
y.bottom.of.vertical.line <- y.location  # store top of vertical line

below.y.bottom.of.vertical.line <- y.bottom.of.vertical.line - small.space/2

if (!draw.gray.background) {
# four optional grid lines may be drawn on the plot parallel to the spine
  if (draw.optional.grid.lines) {
    segments(0.55, y.top.of.vertical.line, 0.55,
      y.bottom.of.vertical.line, col = "black", lty = "solid", lwd = 1)

    segments(0.65, y.top.of.vertical.line, 0.65,
      y.bottom.of.vertical.line, col = "gray", lty = "solid", lwd = 1)

    segments(0.85, y.top.of.vertical.line, 0.85,
      y.bottom.of.vertical.line, col = "gray", lty = "solid", lwd = 1)

    segments(0.95, y.top.of.vertical.line, 0.95,
      y.bottom.of.vertical.line, col = "black", lty = "solid", lwd = 1)
    }
  }
```

```
# gray background for plotting area of the points
if (draw.gray.background) {
  rect(xleft = 0.55, ybottom = y.bottom.of.vertical.line,
    xright = 0.95, ytop = y.top.of.vertical.line, density = -1, angle = 45,
    col = "light gray", border = NULL, lty = "solid", lwd = 1)

# four optional grid lines may be drawn on the plot parallel to the spine
  if (draw.optional.grid.lines) {
    segments(0.55, y.top.of.vertical.line, 0.55,
      y.bottom.of.vertical.line, col = "black", lty = "solid", lwd = 1)

    segments(0.65, y.top.of.vertical.line, 0.65,
      y.bottom.of.vertical.line, col = "white", lty = "solid", lwd = 1)

    segments(0.85, y.top.of.vertical.line, 0.85,
      y.bottom.of.vertical.line, col = "white", lty = "solid", lwd = 1)

    segments(0.95, y.top.of.vertical.line, 0.95,
      y.bottom.of.vertical.line, col = "black", lty = "solid", lwd = 1)
    }
  }
# draw the all-important spine on the plot
segments(x.center.position, y.top.of.vertical.line, x.center.position,
  y.bottom.of.vertical.line, col = "black", lty = "dashed", lwd = 1)

# horizontal line at top
segments(0.55, y.top.of.vertical.line, 0.95, y.top.of.vertical.line,
     col = "black", lty = 1, lwd = 1)

# horizontal line at bottom
segments(0.55, y.bottom.of.vertical.line, 0.95, y.bottom.of.vertical.line,
       col = "black", lty = 1, lwd = 1)

# plot for ticks and labels
segments(0.55, y.bottom.of.vertical.line,
  0.55, below.y.bottom.of.vertical.line,
  col = "black", lty = 1, lwd = 1)   # tick line at bottom

segments(0.65, y.bottom.of.vertical.line,
  0.65, below.y.bottom.of.vertical.line,
  col = "black", lty = 1, lwd = 1)   # tick line at bottom

segments(0.75, y.bottom.of.vertical.line,
  0.75, below.y.bottom.of.vertical.line,
  col = "black", lty = 1, lwd = 1)   # tick line at bottom

segments(0.85, y.bottom.of.vertical.line,
  0.85, below.y.bottom.of.vertical.line,
  col = "black", lty = 1, lwd = 1)   # tick line at bottom

segments(0.95, y.bottom.of.vertical.line,
  0.95, below.y.bottom.of.vertical.line,
  col = "black", lty = 1, lwd = 1)   # tick line at bottom
```

```
# axis labels vary with the max.is.less.than range being used
if (max.is.less.than.1) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-1","-0.5","0","+0.5","+1"),cex=0.75)

if (max.is.less.than.2) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-2","-1","0","+1","+2"),cex=0.75)

if (max.is.less.than.4) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-4","-2","0","+2","+4"),cex=0.75)

if (max.is.less.than.10) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-10","-5","0","+5","+10"),cex=0.75)

if (max.is.less.than.20) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-20","-10","0","+10","+20"),cex=0.75)

if (max.is.less.than.40) text(c(0.55,0.65,0.75,0.85,0.95),
  rep(below.y.bottom.of.vertical.line,times=5),
  pos=1,labels=c("-40","-20","0","+20","+40"),cex=0.75)

y.location <- below.y.bottom.of.vertical.line - small.space

text(.75,y.location,pos=1,labels=c("Part-Worth"),
  cex=0.95)

y.location <- below.y.bottom.of.vertical.line - small.space

text(0.75,y.location,pos=1,labels=c("Part-Worth"), cex=0.95)

if(print.internal.consistency) {
  y.location <- y.location - medium.space
  text(c(0.525),c(y.location),pos=2,labels=paste("Internal consistency: ",
  pretty.print(conjoint.results$internal.consistency),
  sep=""))
  }

# if we have grid lines we may have plotted over part-worth points
# if we have a gray background then we have plotted over part-worth points
# so let us plot those all-important part-worth points and lines once again
if(draw.gray.background || draw.optional.grid.lines) {
  y.location <- y.top.of.vertical.line  # retreive the starting value

# repeat the primary plotting loop
for(k in seq(along=effect.names)) {
  y.location <- y.location - large.space
  text(c(0.4),c(y.location),pos=2,
    labels=paste(effect.name.map(effect.names[k])," ",sep=""),cex=01.0)
```

```
    text(c(0.525),c(y.location),pos=2,col=color.for.printing.importance.text,
      labels=paste(" ",left.side.symbol.to.print.around.importance,
      pretty.print(
      unlist(conjoint.results$attribute.importance[effect.names[k]])),"%",
      right.side.symbol.to.print.around.importance,sep=""),cex=01.0)

 # begin loop for printing part-worths
      for(m in seq(1:number.of.levels.of.attribute[k])) {
         y.location <- y.location - medium.space
         text(c(0.4),c(y.location),pos=2,
         conjoint.results$xlevel[[effect.names[k]]][m],cex=01.0)

         text(c(0.525),c(y.location),
           pos=2,col=color.for.printing.part.worth.text,
           labels=paste(" ",left.side.symbol.to.print.around.part.worths,
           pretty.print(part.worth.plotting.list[[effect.names[k]]][m]),
           right.side.symbol.to.print.around.part.worths,sep=""),cex=01.0)

      points(part.worth.point.position[[effect.names[k]]][m],y.location,
         type = "p", pch = 20, col = color.for.part.worth.point, cex = 2)
      segments(x.center.position, y.location,
      part.worth.point.position[[effect.names[k]]][m], y.location,
         col = color.for.part.worth.line, lty = 1, lwd = 2)
      }
    }
  }
}

# user-defined function for plotting descriptive attribute names
effect.name.map <- function(effect.name) {
  if(effect.name=="price") return("Ticket Price")
  if(effect.name=="seating") return("Seating Area")
  if(effect.name=="boxvip") return("Box/VIP")
  if(effect.name=="frontrow") return("Front Row")
  if(effect.name=="promotion") return("Promotion")
  }
# read in conjoint survey profiles with respondent ranks
conjoint.data.frame <- read.csv("sporting_event_ranking.csv")

# set up sum contrasts for effects coding as needed for conjoint analysis
options(contrasts=c("contr.sum","contr.poly"))
# main effects model specification
main.effects.model <-
    {ranking ~ price + seating + boxvip + frontrow + promotion}

# fit linear regression model using main effects only (no interaction terms)
main.effects.model.fit <- lm(main.effects.model, data=conjoint.data.frame)
print(summary(main.effects.model.fit))

# save key list elements of the fitted model as needed for conjoint measures
conjoint.results <-
  main.effects.model.fit[c("contrasts","xlevels","coefficients")]
conjoint.results$attributes <- names(conjoint.results$contrasts)
```