Diana Bertuol Garcia

Mestranda em Ecologia, Instituto de Biociências, USP

O título de minha dissertação é “Causas da lacuna pesquisa-prática na área ambiental: pontos de vista de pesquisadores e tomadores de decisão no Brasil”, com orientação da Profa. Dra. Renata Pardini.

Meus exercícios

Linque para a página com os meus exercícios resolvidos: exec.

Trabalho Final

Linque para a página com as minhas Propostas de Trabalho Final

Arquivos

Arquivo da função agree.sort
Página de ajuda help-agree.sort
Arquivos de dados para os exemplos Exemplo 1 Exemplo 2 Exemplo 3

Help da função

```
agree.sort                package:nenhum                R Documentation

Non-Metric Multidimensional Scaling de dados provenientes de atividades de
pile sorting

Description:

    A função produz matrizes de similaridade e dissimilaridade item por item
entre as diferentes realizações de uma atividade de pile sorting. A partir
da
matriz de dissimilaridade, realiza uma Non-Metric Multidimensional Scaling
(NMDS) e apresenta os resultados em gráficos que podem ser coloridos de
acordo com a classificação de cada realização do pile sorting. Não realiza
uma NMDS de verdade, mas chama outra função para isso (o default é
monoMDS).
```

```
Usage:

    agree.sort <- function(dados, ign=NA, it.names=NULL, usar=c("monoMDS",
"isoMDS", "metaMDS"), classif=NULL, graph.1 = TRUE, choices=c(1,2), cex=1.2,
pch=16, ...)

Arguments:

    dados   lista de realizações de pile sorting. Cada posição da lista
            deve conter um dataframe com as   categorias nas colunas.

    ign     sequência de caracteres a serem ignorados dentro de cada
            dataframe.

    it.names    nomes dos itens que foram utilizados no pile sorting a serem
            plotados no gráfico. Deve ser um vetor de tamanho igual ao
                número de itens do pile sorting. O default NULL procura os
nomes
                dos itens a partir dos próprios caracteres dentro dos
dataframes.

    usar    função a ser usada para o NMDS. O default é monoMDS do pacote
        vegan, mas podem ser utilizadas as funções isoMDS do pacote
            MASS ou metaMDS do pacote vegan.

    classif realizações do pile sorting a serem utilizadas para colorir
            os gráficos. Deve ser um vetor numérico correspondente às
                posições da lista de dados a serem utilizadas.
    graph.1 vetor lógico indicando se o primeiro gráfico, sem diferenças
                entre classificações, deve ser produzido ou não.

    choices vetor de comprimento 2 indicando as dimensões da NMDS a serem
        utilizadas nos eixos dos gráficos. O default é c(1,2).

    cex     tamanho dos símbolos e textos plotados nos gráficos.

    pch     símbolo para plotagem nos gráficos.

    ...     argumentos opcionais das funções do argumento "usar" e de
            plotagem de gráficos.

Details:
    Em análise de dados qualitativos, pesquisadores costumam identificar
tópicos recorrentes nos dados e criar classificações para estes tópicos. O
pile sorting é uma técnica exploratória utilizada para esse processo e
envolve pedir para pessoas agruparem itens (podem ser trechos de texto,
imagens, objetos, etc) em pilhas de coisas similares e criarem critérios de
classificação para essas pilhas/categorias. A função agree.sort é uma forma
de analisar a da concordância entre as realizações do pile sorting.
    Os passos da função são:
```

```
    1. Construção de matriz de similaridade item por item para cada
realização do pile sorting. O valor alocado para a célula i,j será 1 quando
o
item i e o item j foram colocados na mesma pilha/categoria (mesma coluna do
dataframe) e 0 quando foram colocados em diferentes pilhas.
    2. Construção de matriz de similaridade item por item geral. O valor
da célula i,j será a proporção (de 0 a 1) das realizações de pile sorting em
que os itens i e j foram alocados para uma mesma pilha (mesma coluna do
dataframe). Esse valor será 1 quando os itens i e j apareceram na mesma
pilha
em todas as realizações e 0 quando não foram colocados nenhuma vez na mesma
pilha.
    3. Construção da matriz de dissimilaridade item por item geral. Os
valores dessa matriz são calculados por 1 menos os valores da matriz de
similaridade.
    4. Realização da NMDS com a função determinada no argumento "usar", a
partir da matriz de dissimilaridade.
    5. Construção do gráfico do resultado da NMDS.
    6. Construção de gráficos para cada classificação pedida no argumento
"classif", com os pontos diferenciados de acordo com o que foi determinado
no
argumento.

Value:

    Retorna uma lista, com as seguintes posições:
    itens:    vetor, de caracteres ou numérico, dos itens
          utilizados no pile sorting.

    similaridade: matriz de similaridade entre os itens.

    diferencas:   matriz de dissimilaridade entre os itens.

    mds:      objeto da classe nmds, resultado das funções do argumento
              "usar". Ver monoMDS, isoMDS e metaMDS para mais detalhes.

Warning:

    A função é interrompida e mensagens de erro são retornadas quando o
argumento "dados" não for um objeto da classe lista, as posições da lista
"dados" não forem dataframes ehouver itens repetidos dentro de um mesmo
dataframe.

Notes:

    Os itens dentro de um dataframe da lista "dados" não devem se repetir
entre as colunas. É possível que nem todos os itens estejam presentes em
todos os dataframes.

    O argumento "it.names" é indicado apenas quando os caracteres nos
dataframes da lista de dados não são correspondem aos nomes dos itens. Esse
```

```
argumento é utilizado apenas na produção dos gráficos e não afeta o cálculo
das matrizes.
    A função isoMDS só aceita valores positivos de distância (exceto na
diagonal), portanto não lida bem com os casos em que alguns itens foram
colocados juntos em todas as realizações do pile sorting. Nesses casos,
monoMDS ou metaMDS são mais adequados.
    A função metaMDS também não realiza um NMDS própria, mas utiliza as
funções monoMDS ou isoMDS, realizando-as várias vezes com diferentes
configurações iniciais e optando pela melhor solução (com o menor stress).

    São produzidos gráficos apenas de duas dimensões. Para gráficos de 3
dimensões ver funções dos pacotes lattice ou scatterplot3d.


Author(s):

    Diana Bertuol Garcia
    dia.bertuol@gmail.com

References:

    Para análise de pile sorting, ver:
    http://petergiovannini.com/ethnobotany-methods/how-to-pile-sorting-
with-anthropac-tutorial.html
    Bernard, H.R. and G.W. Ryan 2010. Analyzing Qualitative Data:
Systematic Approaches. Los Angeles: Sage. 451p.

    Weller, S.C and A.K. Romney 1988. Systematic data collection.
California: Sage. 96p.

    Para Non-Metroc Multidimensional Scaling, ver referências em isoMDS,
monoMDS e metaMDS.
See Also:

    Funções isoMDS, monoMDS e metaMDS.

Examples:

#### Exemplo 1: construção do grafico da NMDS.
##atividade de sorting com algumas frutas e vegetais, feita por 6 pessoas.
# lendo os dados
pessoas <- lapply(dir(pattern = "frutas-p"), read.table, header=F, as.is=T,
sep=";")
# averiguando a concordância e similaridade entre frutas/vegetais e entre
classificações
frutas <- agree.sort(dados=pessoas, ign="", usar="isoMDS") ## usando a
função isoMDS e mostrando as classificações de todas as pessoas
## Observe há existência de alguns grupos bem definidos, como o grupo
abobrinha, brócolis, alho poró e berinjela (legumes)
agree.sort(dados=pessoas, ign="", usar="isoMDS", classif=1:5) ## vendo todos
```

```
os gráficos em sequência. Perceba como a classificação 3 tem grupos bem
definidos, pode-se procurar seus critérios para tentar embasar a separação
nesses grupos.
frutas.it <- frutas$itens ## lista de todas as frutas e vegetais

#### Exemplo 2: Exemplo da criação de graficos para cada classificação.
## atividade de sorting com 15 trechos relacionados às causas para a lacuna
pesquisa-prática em conservação, realizada por 6 pessoas.
# lendo os dados
sortings <- lapply(dir(pattern = "Pessoa"), read.table, header=T, as.is=T,
sep=";")
# averiguando a concordância e similaridade entre as causas e entre
classificações
par(mfrow=c(2,4)) ## preparando a janela gráfica para visualizar todos os
gráficos em conjunto
causas <- agree.sort(dados=sortings, usar="monoMDS", classif=c(1:6)) ##
usando a função monoMDS e mostrando as classificações de todas as pessoas
par(mfrow=c(1,1)) ## volta os parâmetros gráficos originais.
## Observe como não há grupos bem definidos de tipos de causas nesses
trechos de texto, nem muita concordância entre os participantes

#### Exemplo 3: uso de metaMDS, definição de nomes de itens nos argumentos
da função e não impressao do primeiro gráfico
## atividade de sorting em 2 pilhas de algumas expressões relacionadas a
questão "o que é amor?".
## entrada de dados
realizacoes <- lapply(dir(pattern = "sent-"), read.table, header=F, as.is=T,
sep=";")
sent <-
c("not_material","sadness","being_together","sex_un","kind","first_stg","hap
py","upsurge","idealize_wrld","altruism","giddy","tender","disapt","stupid",
"surprise","mutual","friend","not_ord","temp","care","butterfly") ##
definição dos nome das expressoes
x11() ## abre janela gráfica
par(mfrow=c(2,3)) ## espaço para 6 gráficos
sentimentos <- agree.sort(dados=realizacoes, it.names=sent, usar="metaMDS",
classif=1:5) ## usando a função metaMDS, definindo nomes dos itens
(diferentes do que se encontram na lista de dados) e mostrando metade das
classificações.
par(mfrow=c(1,1)) ## volta os parâmetros gráficos aos originais
x11() ## abre outra janela gráfica
par(mfrow=c(2,3)) ## espaço apra 6 gráficos
agree.sort(realizacoes, graph.1=F, it.names=sent, usar="metaMDS",
classif=6:10)## mostrando a outra metade das classificações e sem produzir o
primeiro gráfico.
par(mfrow=c(1,1)) ## volta os parâmetros gráficos aos originais
### Observe o agrupamento dos aspectos mais positivos relacionados ao amor
(happy, friend, upsurge, altruism, being/-together, kind) e o agrupamento
dos aspectos mais negativos (stupid, sadness, idealize-world).
```

Código da função

```
##########################################################################
########################
################################### Função agree.sort
#############################################
##########################################################################
########################

####para analisar concordância (agreement) entre atividades de sortings de
diferentes pessoas e similaridade entre os itens

agree.sort <- function(dados, ign=NA, it.names=NULL, usar=c("monoMDS",
"isoMDS", "metaMDS"), classif=NULL, graph.1 = TRUE, choices=c(1,2), cex=1.2,
pch=16, ...){ ## define o nome da função e os argumentos, colocando alguns
default para os gráficos. Dados = lista de dataframes com realizações dos
sort, ign é o caracter que vai ser ignorado mais pra frente, it.names são os
nomes dos itens, usar é a função a ser usada, classif é qual realizações de
sorting usar para colorir os gráficos e os outros são opções gráficas.
  usar <- match.arg(usar) ## define as opções para o argumento "usar",
deixando o primeiro como default e dando msg de erro se der opção que não
existe
##########################################################################
########################
  ######################### verificando os argumentos
################################################
  #### verificando se o objeto de entrada de dados está correto
  if(class(dados)!="list"){ ## se o objeto não for uma lista
    stop("Objeto de entrada de dados deve ser uma lista") ## para a função e
imprime uma msg de erro na tela
  } ## fecha o if
  for(p in 1:length(dados)) { ## para cada posição da lista p, ou seja, p é
cada dataframe
    if(is.data.frame(dados[[p]])==F){ ## se não foram dataframes na lista
      stop("As posições da lista 'dados' devem ser dataframes") ## para a
função e imprime uma msg de erro
    } ##fecha o if
  } ## fecha o ciclo
  #### verificando se os outros argumento estao correto
  if (length(choices)!=2) { ## se o comprimento for difernete de 2
    stop("O argumento choices deve ter comprimento 2") ## para a função e
imprim msg de aviso
  } ## fecha o if
##########################################################################
########################
  ################ montando um tipo de lista padrão, com NA onde tem que
ignorar ####################
  if(!is.na(ign)){ ## para os casos em que foi pedido para ignorar algo que
não é NA
```

```
    for (p in 1:length(dados)) { ## o p vai ser todas as posições da lista.
Ou seja, uma realização do sort de cada vez
      for (c in 1:dim(dados[[p]])[2]){ ### c vai ser cada coluna dos
dataframes. Para a posição p (ver acima) do objeto dados, pega o segundo
valor das dimensões, então o número de colunas do dataframe.
        dados[[p]][dados[[p]][c]==ign,c] <- rep(NA,
times=sum(dados[[p]][c]==ign)) ## para cada coluna de cada dataframe,
substitui por NA (repetido quantas vezes houver o caracter de ign) todos os
locais em que houver o caracter de ign.
      } ## fecha o ciclo das colunas do dataframe
    } ## fecha o ciclo dos dataframes
  } ## fecha o  if
##############################################################################
########################
  ############### criando, a partir dos dados, a lista de todos os itens que
foram alocados.#########
  itens <- NULL ## cria o objeto itens para ser completado no for abaixo
  for (p in 1:length(dados)) { ## o p vai ser todas as posições da lista. Ou
seja, uma realização do sort de cada vez
    for (c in 1:dim(dados[[p]])[2]){ ### c vai ser cada coluna dos
dataframes, ou seja, as pessoas. Para a posição p (ver acima) do objeto
dados, pega o segundo valor das dimensões, isto é, o número de colunas do
dataframe.
      itens.mom <- dados[[p]][,c] ## objeto itens.mom vai ter,
momentaneamente, o que está escrito em cada coluna de cada dataframe
      itens <- c(itens, itens.mom) ## cria o objeto de itens concatenando
todas as colunas de todos os dataframes
    } ## fecha o ciclo de coluna
  } ## fecha o ciclo de posição na lista do objeto dados
  itens <- unique(itens) ## retira as repetições do vetor itens.mom
  itens <- itens[order(itens, na.last=NA)] ## ordena do menor para o maior
ou em ordem alfabétia, excluindo-se os NAs
  if(is.null(it.names)==FALSE){ ## caso o argumento it.names tenha sido
fornecido (e não seja nulo)
    names(itens) <- it.names ## nomeia os itens da matriz com os nomes
fornecidos pelo argumento it.names
  } ## fim do if
  else { names(itens) <- itens } ## se o argumento itens não for dado
(default é NULL), pega do que está na matriz mesmo
##############################################################################
########################
  ################################## checando se tem repetição de itens
##############################
  for (p in 1:length(dados)) { ## o p vai ser todas as posições da lista. Ou
seja, uma realização do sort de cada vez
    for (i in itens) { ## i vai ser cada item
      item.i <- dados[[p]] == i ## o vetor lógico de onde está os itens
      if (sum(item.i, na.rm=T)>1) { ## se houver mais de uma vez o item
        stop(paste(paste(paste("Os itens não devem se repetir dentro de cada
realização do pile sorting. Reveja o item", i, sep=" "), p, sep=" na "), "a
realização do pile sorting", sep="")) ## pára a função e dá um aviso para
```

```
rever o que está errado.
      } ## fecha o if
    } ## fecha o ciclo dos itens
  } ## fecha o ciclo das posições da lista
##############################################################################
######################
  ################################### cálculos das matrizes
########################################
  #### matrizes de similaridade para cada pessoa/dataframe
  matrizes.ind <- array(NA, dim=c(length(itens), length(itens),
length(dados))) ## cria o objeto array para colocar as matrizes depois. As
duas primerias dimensões são a matriz (item por item) e a terceira são as
pessoas
  for(p in 1:length(dados)){ ## para cada posição da lista p, ou seja, p é
cada pessoa/dataframe.
    matriz <- matrix(NA, nrow=length(itens), ncol=length(itens)) ## cria uma
matriz para a pessoa
    for (i in 1:(length(itens)-1)){ ## i varia do primeiro ao último item.
Por isso vai todos, menos o último, pq quando chegar no último, ele já foi
comparado com todos antes
      for (j in 2:length(itens)){ ## J vai ser os próximos itens para serem
comparados com o i acima.
        co.ij <- dados[[p]]==itens[i]|dados[[p]]==itens[j] ## monta um
dataframe lógico, onde os TRUES aparecem quando tem o item j ou o item i
        co.ij.2 <- apply(co.ij, MARGIN=2, FUN=sum, na.rm=T) ## soma por
coluna o dataframe lógico
        if (sum(co.ij.2==2)==1){ ## se a soma de alguma das colunas for 2
(ou seja, os dois itens aparecem na mesma categoria/coluna).
          matriz[i,j] <- 1 ## coloca o valor 1 na cédula i,j, que quer dizer
q estão na mesma categoria
          matriz[j,i] <- 1 ## coloca o valor 1 na cédula j,i. Idem acima.
        } ## fecha o if
        else{ ## se não for 2, ou seja, se eles não aparecerem na mesma
coluna
          matriz[i,j] <- 0 ## põe o valor 0 na célula i,j, que quer dizer
que não estão na mesma categoria
          matriz[j,i] <- 0 ## põe o valor 0 na célula j,i. Idem acima.
        } ## fecha o else
      } ## fecha o ciclo do j
    } ## fecha o ciclo do i
    diag(matriz) <- 1 ## a diagonal da matriz é um (pois um item sempre está
na mesma categoria que ele mesmo)
    matrizes.ind[,,p] <- matriz ## coloca a matriz da pessoa no array das
categorias
    dimnames(matrizes.ind) <- list(names(itens), names(itens), NULL) ## dá
nome às primeiras dimensões do array (que sao as dimensões das matrizes
individuais)
  } ## fecha o ciclo do p
  #### matriz geral de similaridade
  matriz.S.ger <- apply(matrizes.ind, c(1,2), mean) ## calcula a matriz de
```

```
similaridade  entre todas as pessoas ao fazer a proporção de vezes em que o
1 aparece entre dois itens. Faz a média, pois a média soma tudo e divide
pelo total de números, sendo a mesmo conta que a proporção nesse caso
(quantas vezes ocorre o valor 1, dividido pelo total de vezes)
  #### matriz geral de dissimilaridade
  matriz.D.ger <- 1-matriz.S.ger ## inverte os valores para ficar com matriz
de dissimilaridade geral
##############################################################################
#######################
  ################################## Non-Metric Multidimensional Scaling
############################
  if (usar=="monoMDS"){ ## quando a opção de função a ser usada é monoMDS
    require("vegan") ## instala o pacote vegan, que contém a função monoMDS
    mds <- monoMDS(matriz.D.ger,...) ## faz o MDS com a função monoMDS
  } ## fecha o if
  if (usar=="isoMDS"){ ## quando a opção de função a ser usada é isoMDS
    require("vegan")
    library("MASS") ## instala o pacote MASS, que contém a função isoMDS
    mds <- isoMDS(matriz.D.ger,...) ## faz o MDS com a função isoMDS. OBS.:
só aceita valores positivos de distância!! Mas a própria função já tem a msg
de erro!
    class(mds) <- "nmds" ## transforma o resultado, que era uma lista, em um
objeto do tipo MDS.
  } ## fecha o if
  if (usar=="metaMDS"){ ## quando a opção de função a ser usada é metaMDS
    require("vegan") ## instala o pacote vegan, que contém a função monoMDS
    mds <- metaMDS(matriz.D.ger,...) ## faz o MDS com a função metaMDS.
OBS.: na verdade usa monoMDS como default, mas tenta procurar uma solução
estável. Pode usar isoMDS...
  } ## fecha o if
##############################################################################
#######################
  ################################## colocando os pontos no gráfico
##################################
  ################ antes de tudo, cria o objeto com os scores
############################
  scores <- scores(mds)[,choices] ## cria um objeto com os scores do mds
para as dimensões escolhidas para plotar. OBS.: a função só plota gráficos
de 2 dimensões!!!
  ######## prepara os valores mínimos e máximos dos eixos que serão
colocados nos graficos ##########
  eixos <- matrix(NA, ncol= 2, nrow=3, dimnames=list(c("min", "max", "entre
ponto e nome"), c("x", "y"))) ## cria a matriz onde vou por os valores de
min e max para cada eixo
  for (d in 1:2){ ## para cada dimensão d do MDS
    amplitude <- max(scores[,d])-min(scores[,d]) ## calcula a amplitude dos
valores
    extra <- amplitude/6 ## calcula um sexto da amplitue
    eixos[1,d] <- min(scores[,d])-extra ## valor mínimo vai ser um sexto pra
baixo do mínino, já põe no objeto certo
    eixos[2,d] <- max(scores[,d])+extra ## valor máximo vai ser um sexto pra
```

```
cima do máximo, já pôs no objeto certo
    eixos[3,d] <- amplitude/20 ## valor que vai distanciar o ponto do seu
nome nos gráficos
  } ## fecha o ciclo das dimensões
  ######################## sempre faz o gráfico sem pintar nada diferente
############################
  if(graph.1){
    plot(scores, xlim=eixos[1:2,1], ylim=eixos[1:2,2], main="Non-Metric
Multidimensional Scaling Results", pch=pch, cex=cex, col="black", ...) ##
plota os scores do MDS em um gráfico, com os valores max e min dos eixos
conforme difinidos acima
    text(scores-eixos[3,], labels=names(itens), col="black", ...) ## plota
os nomes dos itens
    text(x=eixos[1,1]+3*eixos[3,1], y=eixos[2,2]-eixos[3,2],
labels=paste("Stress=", round(mds$stress, digits=4), sep=""), col="black",
cex=cex, ...) ## adiciona o valor do stress aos graficos, no ponto mínimo de
x e máximo de y + 3 vezes o um sexto da amplitude... É só uma posição boa de
plotar o valor do stress.
  }
  par(ask=T) ## pede para perguntar antes de por o próximo grafico
  ############################## se pedir classificação,
###############################################
  ###vai fazer um gráfico para cada classificação, pintando as categorias de
cada classificação/pessoa
  if(is.null(classif)==FALSE) { ## se o argumento classif for diferente de
nulo (ou seja, pediu para classificar).
    #### cria os vetores/fatores de categorias para os itens. Preparação dos
dados para plotar os gráficos depois as cores diferentes
    categorias <- matrix(NA, nrow=length(itens), ncol=length(classif)) ##
cria a matriz (a ser preenchida) de itens nas linhas por categorias de cada
pessoa nas colunas
    for (n in 1:length(classif)){ ## n é cada coluna da matriz a ser
preenchida, ou de 1 ao número de classificação/gráficos pedidos no argumento
classif
      for (o in classif[n]){ ## o são os valores das posições pedidas para
classificação (pois não necessariamente o argumento classif vai ter posições
em sequência)
        categorias.o <- rep(NA, times=length(itens)) ## faz o vetor a ser
preenchido pela classificação da pessoa o
        for (i in 1:length(itens)){ ## i é cada item
          posicao.item <- which(dados[[o]]==itens[i]) ## pega a posição em
que o item i está
          coluna.item <- ceiling(posicao.item/dim(dados[[o]])[1]) ## divide
pelo número de linhas e arredonda para cima parag pegar o número da coluna,
que será o nome da categorias em que está colocado
          if (length(coluna.item)!=0) { ## se o valor da categorias for
diferente de zero, ou seja, o item foi alocado em alguma catgeoria pela
pessoa (em contraposição a casos em que o item não aparece no sorting)
            categorias.o[i] <- coluna.item ## aloca o nome da categoria para
a posição correspondente ao item i no vetor das categorias
```

```
          } ## fecha o if. Se o valor for 0, não vai colocar nada,
permanecendo o NA do vetor criado no início.
        } ## fecha o ciclo dos itens i
      } ## fecha o ciclo das posições o
      categorias[,n] <- categorias.o ## aloca para a sua respectiva posição
n na matriz o vetor de categorias do dataframe
    } ## fecha o ciclo do n
    #### plotando um gráfico para cada classificação
    for (n in 1:length(classif)){ ## n é cada coluna da matriz com as
categorias
      fora <- which(is.na(categorias[,n])) ## pega a posição dos NAs (caso
os itens não apreceçam na realização do sorting)
      scores.n <- scores ## cria um objeto igual aos scores, apra poder
modificar algumas linhas dps
      itens.n <- itens ## cria um objeto igual aos itens, apra poder
modificar algumas linhas depois
      if(length(fora)!=0){ ## se houver algo que fica de fora (se não, tem
todos os itens, segue reto)
        scores.n[fora,] <- NA ## põe NA na linha do item que não apreceu na
realização do sorting
        itens.n[fora] <- NA ## põe NA na linha do item que não apareceu na
realização do sorting
        names(itens.n)[fora] <- NA ## põe NA no nome do item que não
apareceu na realização do sorting
      } ## fecha o if dos itens que não estão em todas as categorias
      plot(scores.n, xlim=eixos[1:2,1], ylim=eixos[1:2,2], main=paste("Non-
Metric Multidimensional Scaling Results\n Classificação", classif[n]),
cex=cex, pch=pch, col=categorias[,n], ...) ## plota os scores, com cores
diferentes para cada categoria. Eixos maximo e minino idem gráfico anterior
      text(scores-eixos[3,], labels=names(itens.n), col=categorias[,n], ...)
## plota os nomes dos textos, com as mesmas cores de cima
      text(x=eixos[1,1]+3*eixos[3,1], y=eixos[2,2]-eixos[3,2],
labels=paste("Stress=", round(mds$stress, digits=4), sep=""), col="black",
cex=cex, ...)## adiciona o valor do stress aos graficos, no ponto mínimo de
x e máximo de y
    } ## fecha o ciclo das colunas (n)
  } ## fecha o if da classificação
  par(ask=F) ##retorna os parâmetros gráficos aos originais
##############################################################################
#######################
  return(list(itens=names(itens), similaridade=matriz.S.ger,
diferencas=matriz.D.ger, mds=mds)) ## retorna uma lista com os itens, a
matriz de similaridade geral, a matriz de dissimilaridade geral e o objeto
da MDS (classe nmds)
} ## fim da função agree.sort

##############################################################################
#########################
##############################################################################
#########################
##############################################################################
```

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