João Menezes

Sou aluno de mestrado do Programa de Pós-graduação em Ecologia do IB-USP. Sob orientação do Prof. Eduardo Santos, investigo a influência de fatores ambientais sobre a evolução de displays sexuais em aves.

Meus exercícios

Os scripts dos exercícios postados na tabela se encontram neste link.

Propostas de trabalho final

Estruturação e organização das propostas baseadas em rld (2016).

Proposta A: poker.prob()

Tarefa a ser executada

1. Estima as chances de um jogador vencer uma rodada de pôquer Texas Hold'em, cujas regras podem ser encontradas neste link.
2. Estima as chances de o jogador realizar cada tipo de jogo ao final da rodada.

Ambas as estimativas serão obtidas simulando-se partidas de pôquer, e (1) calculando-se a porcentagem dessas partidas vencidas pelo usuário, e (2) calculando-se as porcentagens de partidas em que cada tipo de jogo foi obtido.

Utilidade da função

Embasa a difícil decisão tricotômica a ser tomada ao se receber as cartas em uma partida de Texas Hold'em: *call* (paga a aposta), *fold* (larga as cartas) ou *raise* (aumenta a aposta)?

Argumentos de entrada

A função possui quatro argumentos de entrada:

- *carta1* e *carta2*: cartas que o usuário possui e para as quais as chances de vencer serão determinadas. Tratam-se de *strings* com dois caracteres concatenados: um representando o valor de cada carta, e outro sendo a primeira letra do respectivo naipe (e.g., "Ac", "9e", "2o", "Kp"). Há, portanto, 52 valores possíveis (i.e., um baralho completo) para cada um destes

argumentos.
- *nadv*: número inteiro representando o número de adversários contra o qual o usuário está jogando.
- *nsim*: número de simulações de jogos, com padrão a ser definido por mim.

Saída da função

A função retornará um objeto do tipo *list* com as seguintes informações:

- a porcentagem de partidas simuladas que foram vencidas com as cartas do usuário. Isso pode ser retornado em forma de vetor (com o valor da porcentagem, apenas), ou em forma de *string* com, por exemplo, a seguinte frase: "X% de chance de vencer *nadv* adversário(s)."
- dentre as simulações, com qual frequência as cartas do usuário resultaram nos diferentes tipos de jogos: carta alta (ausência de jogo), par, dois pares, trio, sequência, *flush*, *full house*, quadra e *straight flush*. Esse objeto é do tipo *data frame*.

> E aí João? Tudo certo?
>
> Parabéns pela organização, escolheu o modelo certo para agradar o monitor haha!
>
> Sua proposta A está muito interessante e muito bem explicada, talvez só um pouco específica demais (não sei qual a proporção da humanidade que joga pôquer E usa R, mas deve ser baixa =P), mas parece que você terá um desafio bom e divertido para criá-la, com alguns FOR e IF pelo caminho.
>
> Mas como eu não entendo nada de pôquer, preciso te fazer umas perguntas antes de decidir qualquer coisa. Até li o link explicativo que você colocou (obrigado!), mas ainda tenho algumas dúvidas sobre o jogo. Sua função serviria para um momento bem específico de cada rodada, na hora que o jogador recebe as cartas, e não para o jogo todo, certo? O jogador vê as cartas, coloca na função e ela calcula o que ele deve fazer, é isso? E quais são os cálculos envolvidos? Apenas ver onde essa combinação sorteada se insere numa hierarquia de todas as combinações possíveis para uma mão? Você já tem esse dado pronto (todas as combinações possíveis de cartas, e quais são melhores do que quais)? Se sim, o que a função fará é calcular a chance de ganhar n vezes e mostrar a probabilidade, de com aquela mão, você ganhar de x adversários? Entendi certo?
>
> Abraços! — *Rodolfo Liporoni Dias 2017/06/02 14:50*

Fala Rodolfo, tudo certo?

Olha só! Ainda bem que dei os créditos, hein? Haha

Muita gente joga pôquer e muita gente usa a R, mas não tenho a mínima ideia de como é a intersecção, haha. No início, imaginei que essa função seria apenas uma boa maneira de fixar os conhecimentos adquiridos na disciplina, mas não muito mais que isso, já que é bem improvável que alguém jogue pôquer com um notebook na mesa. Depois, porém, lembrei da existência e da popularidade do pôquer online, em que a pessoa poderia facilmente jogar e usar R ao mesmo tempo. Arrisco dizer que essa modalidade de pôquer é muito mais jogada do que a "real". Pensando assim, acho que a função tem um potencial legal de aplicação!

O que a função faria, em termos práticos, é:

1. Simular as cartas dos adversários e da mesa usando sample().
2. Interpretar qual a melhor mão de cada jogador (usuário e *nadv* adversários): isso envolve escolher qual a melhor combinação de cinco cartas dentre as sete (duas da mão + cinco da mesa) de cada jogador. Já comecei a desenvolver esse código (vários *if*s) e, apesar de ser a parte mais desafiadora, acredito que conseguirei dar conta.
3. Confrontar as mãos dos jogadores e definir o vencedor.
4. Repetir os passos acima *nsim* vezes e calcular: (a) a porcentagem de rodadas que foram vencidas pelo usuário, (b) a porcentagem de rodadas que o usuário terminou com cada tipo de jogo.

Quanto ao objetivo, você entendeu certinho! Ela poderia ser usada apenas na primeira rodada de apostas, logo após o jogador receber as cartas (essa etapa é conhecida como *pre-flop*). Mas me parece bem possível estender e generalizar a função para as rodadas subsequentes de apostas. A única diferença é que, quanto mais à frente na rodada, mais cartas (i.e., as cartas da mesa) teriam que ser dadas como argumentos pelo usuário ao invés de simuladas pela função. Que acha de eu desenvolver a função para a primeira rodada inicialmente, e, se tiver tempo, eu tento generalizar para outras rodadas?

Espero que tenha esclarecido suas dúvidas.

Abraço! — *João Carnio Teles de Menezes 2017/06/02 20:34*

> Oi João, obrigado pelos esclarecimentos! Já ficou bem mais claro.
>
> Então, acho que pode ser uma função divertida; ter aplicação prática é o de menos, o importante é te trazer desafios, e isso já vi que você vai ter e poderá aprender muitas coisas e sedimentar outras.
>
> Apoio sua ideia de entregar a função para uma única rodada do jogo, e, como um bônus, se conseguir, fazer para mais rodadas. Fica a seu critério.
>
> Você prefere fazer a proposta A então? (Veja meus comentários na outra proposta e pode responder só uma vez!)
>
> Abraços! — *Rodolfo Liporoni Dias 2017/06/03 12:07*

> Oi Rodolfo,
>
> Combinado então! Vou fazer a proposta A e, se der, tento generalizá-la.
>
> Abraço e obrigado pela ajuda! — *João Carnio Teles de Menezes 2017/06/03 14:21*

Proposta B: avifauna()

Tarefa a ser executada

A partir de uma lista de espécies de aves observadas em determinado local, cria um arquivo PDF com essa lista, acompanhada de outras informações pertinentes.

Utilidade da função

Gerar listas condensadas de avifauna, seja para fins pessoais (e.g., listas de *birders*), acadêmicos ou profissionais (e.g., consultoria).

Argumentos de entrada

- O argumento *lista* deve ser um vetor com os nomes das espécies que comporão a lista condensada.
- O argumento *ordem* determina qual ordenação será adotada para a lista final. Ele deve ser um *string* com um dos seguintes valores: “taxonomica” ou “alfabetica”.
- Os argumentos *en, pt, amgl, ambr, amsp, ampr* e *ammg* são lógicos e padronizados para TRUE. Eles se referem à adição de colunas extras de informação à lista final (ver abaixo).

Saída da função

Arquivo PDF contendo uma tabela com as espécies ordenadas de forma apropriada, e com as seguintes colunas extras: nome em inglês, nome em português, nível de ameaça global, nível de ameaça nacional, e níveis de ameaça estaduais (SP, PR, MG).

> E aí João?
>
> Bom, sua proposta B também está bem clara, obrigado! Mas ela é bem mais simples né? Parece que só envolve um acréscimo de itens a uma informação pré-fornecida. Fiquei com uma dúvida: de onde você vai tirar todas essas informações sobre cada espécie de ave? Existe um banco de dados online? E quando a pessoa entrar com uma espécie nova, o que a função vai fazer?
>
> Abraços! — *Rodolfo Liporoni Dias 2017/06/02 15:10*

> Oi Rodolfo!
>
> Seria mais ou menos isso, mesmo. O legal dela seria gerar uma lista com informações essenciais, e pronta para ser usada, por exemplo, em relatórios. Todas as informações estão disponíveis online, mas separadamente. Há algum tempo montei o banco de dados com todas elas juntas, para usar no Excel mesmo (daí veio a ideia dessa proposta).
>
> Quando a pessoa entrasse com uma espécie que não está na lista oficial de aves do Brasil, retornaria alguma mensagem de erro do estilo: “A espécie “X y” não consta na lista do CBRO”.
>
> Se tiver mais dúvidas, fique à vontade para perguntar.
>
> Abraço! — *João Carnio Teles de Menezes 2017/06/02 20:50*

> Oi de novo!
>
> Entendi seu ponto e a ideia parece bem redondinha e factível e com uma utilidade maior que a proposta A, porém ainda me parece menos desafiadora que a proposta A. Do ponto de visto do aprendizado, recomendo fazer a proposta A.
>
> Topa? Se sim, já pode mandar bala no código final...
>
> Abraços! — *Rodolfo Liporoni Dias* *2017/06/03 12:11*

Trabalho final: função prob.poker()

Código da função

Arquivo do código: funcao_prob.poker.r

```
#### FUNCAO PROB.POKER ####

prob.poker <-
function(mao1,mao2,flop1=NULL,flop2=NULL,flop3=NULL,turn=NULL,river=NULL,nad
v,nsim=1000)
{
  cat("Calculando chances da mão",mao1,"e",mao2,"contra",nadv,"adversário(s)
em",nsim,"simulações, com as seguintes cartas na
mesa:",flop1,flop2,flop3,turn,river,"\n\n") # mensagem para relembrar os
argumentos inseridos
  cartas.arg <- c(mao1,mao2,flop1,flop2,flop3,turn,river) # vetor com todas
as cartas que foram dadas como argumento (NULLs somem)
  #### criando o baralho de cartas virtuais ####
  numero <- c(rep(2:10,each=4),rep(c("J","Q","K","A"),each=4)) # vetor com
os valores de todas as cartas
  naipe <- rep(c("p","c","o","e"),times=13) # vetor com os naipes de todas
as cartas
  baralho <- data.frame(carta=paste(numero,naipe,sep=""),numero,naipe) #
data frame com tres colunas: numero, naipe, e concatenacao dos dois
  baralho$numero <-
factor(x=baralho$numero,levels=c("2","3","4","5","6","7","8","9","10","J","Q
","K","A")) # transformando a coluna numero em fator
  baralho$carta <- as.character(baralho$carta) # transformando carta em
caractere, caso contrario sample() retirava cartas do baralho e transformava
em numeros
```

```
  baralho$ordem.num1 <- rep(2:14,each=4) # criando uma coluna com a primeira
ordenacao possivel, em que A tem valor maximo
  baralho$ordem.num2 <- c(rep(2:13,each=4),rep(1,each=4)) # criando uma
coluna com a segunda ordenacao possivel, em que A tem valor minimo
  #### testes de premissa ####
  if(length(setdiff(cartas.arg,baralho$carta))>0) # se alguma carta dada
como argumento não pertencer ao conjunto de dados do baralho...
  {
    stop("A seguinte carta não é válida:
",setdiff(cartas.arg,baralho$carta)[1],"\n  Lembre-se: a função diferencia
maiúsculas de minúsculas") #... a funcao eh interrompida e avisa o usuario
  }
  if(max(table(cartas.arg))!=1) # se alguma carta foi inserida mais de uma
vez nos argumentos
  {
    stop("A seguinte carta foi inserida mais de uma vez como argumento:
",names(table(cartas.arg)[table(cartas.arg)>1])[1],"\n  Lembre-se: pôquer é
jogado com apenas um baralho") #... a funcao eh interrompida e avisa o
usuario
  }
  if(nadv>22) # se o numero de adversarios for superior ao que a quantidade
de cartas do baralho permite (7 + 2*nadv tem que ser ≤ a 52)...
  {
    stop("Muitos adversários estão jogando\n  Lembre-se: pôquer não é
coração de mãe") # a funcao eh interrompida e avisa o usuario
  }
  #### funcao que analisa as 7 cartas de um jogador (mao + mesa) e retorna a
melhor combinação de 5 possível ####
  aval.cartas <- function(valores,naipes) # argumento "valores" recebe um
vetor de 7 numeros com os valores das cartas, ordenados de forma crescente.
argumento naipes recebe vetor de 7 caracteres com a primeira letra dos
naipes. esses vetores são gerados ja no formato correto dentro da funcao
prob.poker. o vetor de saida tambem eh gerado no formato certo para ser
interpretado pela funcao prob.poker
  {
    # a funcao avaliara se a mao corresponde a cada tipo de jogo, do mais
forte para o mais fraco, realizando testes logicos para cada tipo de jogo
(ifs)
    ### STRAIGHT FLUSH = cinco cartas com valor em sequencia e do mesmo
naipe ###
    if(max(table(naipes))>=5) # um naipe aparece pelo menos 5 vezes (flush)?
    {
      naipe.flush <- names(table(naipes))[table(naipes)>=5] # nome do naipe
que aparece 5 vezes
    valores.flush <- valores[naipes==naipe.flush] # valores que tem esse
naipe
    if((sum(na.rm=T,diff(valores.flush[1:5])==1)>=4 &
max(na.rm=T,diff(valores.flush[1:5]))==1) |
(sum(na.rm=T,diff(valores.flush[1:6])==1)>=4 &
max(na.rm=T,diff(valores.flush[1:6]))==1) |
(sum(na.rm=T,diff(valores.flush[1:7])==1)>=4 &
```

```
max(na.rm=T,diff(valores.flush[1:7]))==1) |
(sum(na.rm=T,diff(valores.flush[2:6])==1)>=4 &
max(na.rm=T,diff(valores.flush[2:6]))==1) |
(sum(na.rm=T,diff(valores.flush[2:7])==1)>=4 &
max(na.rm=T,diff(valores.flush[2:7]))==1) |
(sum(na.rm=T,diff(valores.flush[3:7])==1)>=4 &
max(na.rm=T,diff(valores.flush[3:7]))==1)) # algum dos possiveis intervalos
de cinco-sete valores do mesmo naipe tem que ter 4 diffs iguais a 1 entre os
valores, e nenhuma das diffs pode ser maior do que 1 - o criterio de diff so
funciona porque os valores ja sao dados ordenados no argumento
    {
      desstfl.temp <- rep(NA,6) # vetor em que estarao os possiveis valores
de desempate
      if(sum(na.rm=T,diff(valores.flush[1:5])==1)>=4 &
max(na.rm=T,diff(valores.flush[1:5]))==1) # se sequencia estiver entre as
posicoes 1 e 5
      {
        desstfl.temp[1] <- valores.flush[5] # valor de desempate (maior da
sequencia) esta na posicao 5
      }
      if(sum(na.rm=T,diff(valores.flush[1:6])==1)>=4 &
max(na.rm=T,diff(valores.flush[1:6]))==1) # se entre 1 e 6
      {
        desstfl.temp[2] <- valores.flush[6] # na posicao 6
      }
      if(sum(na.rm=T,diff(valores.flush[1:7])==1)>=4 &
max(na.rm=T,diff(valores.flush[1:7]))==1) # se entre 1 e 7
      {
        desstfl.temp[3] <- valores.flush[7] # na posicao 7
      }
      if(sum(na.rm=T,diff(valores.flush[2:6])==1)>=4 &
max(na.rm=T,diff(valores.flush[2:6]))==1) # se entre 2 e 6
      {
        desstfl.temp[4] <- valores.flush[6] # na posicao 6
      }
      if(sum(na.rm=T,diff(valores.flush[2:7])==1)>=4 &
max(na.rm=T,diff(valores.flush[2:7]))==1) # se entre 2 e 7
      {
        desstfl.temp[5] <- valores.flush[7] # na posicao 7
      }
      if(sum(na.rm=T,diff(valores.flush[3:7])==1)>=4 &
max(na.rm=T,diff(valores.flush[3:7]))==1) # se entre 3 e 7
      {
        desstfl.temp[6] <- valores.flush[7] # na posicao 7
      }
        desstfl <- max(na.rm=T,desstfl.temp) # valor de desempate (i.e.,
para caso dois jogadores executem o mesmo tipo de jogo). no caso do straight
flush, o desempate é o valor maximo da sequencia, ou seja, o valor maximo
encontrado acima
return(c(jogo=9,desemp1=desstfl,desemp2=0,desemp3=0,desemp4=0,desemp5=0)) #
```

```
retorna o valor do tipo de jogo "straight flush" (9, maior possivel) e o
valor do desempate (outros criterios de desempate nao se aplicam, porque nao
é possivel haver dois straight flush com a mesma carta maxima; tais
criterios retornam "0")
    }
    }
    ### QUADRA = quatro cartas de um mesmo valor ###
    if(max(table(valores))==4) # algum valor de carta aparece quatro vezes?
    {
      desquad1 <- names(table(valores)[table(valores)==4]) # desempate1:
qual valor aparece quatro vezes
      desquad2 <- max(setdiff(valores,desquad1)) # desempate2: carta mais
alta que sobrou (so se aplica se a quadra inteira aparecer na mesa, ja que
so ha 4 cartas de mesmo valor em um baralho)
return(c(jogo=8,desemp1=desquad1,desemp2=desquad2,desemp3=0,desemp4=0,desemp
5=0)) # retorna valor do tipo de jogo (8) e dos desempates
    }
    ### FULL HOUSE = tres cartas de um mesmo valor, duas de outro ###
    if(sort(table(valores),decreasing=T)[1]==3 # o primeiro valor mais
frequente aparece tres vezes?
       & sort(table(valores),decreasing=T)[2]>=2) # o segundo valor mais
frequente aparece duas ou mais vezes? ("ou mais" porque dois trios equivalem
a um trio e uma dupla (full house) ja que o limite de cartas eh 5)
    {
      desfuho1 <- max(as.numeric(names(table(valores)[table(valores)==3])))
# desempate1: maior valor que aparece tres vezes
      desfuho2 <-
max(setdiff(as.numeric(names(table(valores)[table(valores)>=2])),desfuho1))
# desempate2: maior valor que aparece duas ou mais vezes e que eh diferente
do valor acima
return(c(jogo=7,desemp1=desfuho1,desemp2=desfuho2,desemp3=0,desemp4=0,desemp
5=0)) # retorna valor do tipo de jogo (7) e dos desempates
    }
    ### FLUSH = cinco cartas do mesmo naipe ###
    if(max(table(naipes))>=5) # um naipe aparece pelo menos cinco vezes?
    {
      naipe.flush <- names(table(naipes))[table(naipes)>=5] # nome do naipe
que aparece 5 vezes
      desflu1=max(valores[naipes==naipe.flush]) # desempate1: valor da maior
carta desse naipe
      desflu2=max(setdiff(valores[naipes==naipe.flush],desflu1)) #
desempate2: valor da maior sem contar a de cima
      desflu3=max(setdiff(valores[naipes==naipe.flush],c(desflu1,desflu2)))
# desempate3: maior sem contar as de cima
desflu4=max(setdiff(valores[naipes==naipe.flush],c(desflu1,desflu2,desflu3))
) # desempate4: idem
desflu5=max(setdiff(valores[naipes==naipe.flush],c(desflu1,desflu2,desflu3,d
esflu4))) # desempate5: idem
return(c(jogo=6,desemp1=desflu1,desemp2=desflu2,desemp3=desflu3,desemp4=desf
lu4,desemp5=desflu5)) # retorna valor do tipo de jogo (6) e dos desempates
    }
```

```
    ### SEQUENCIA = cinco cartas com valor em sequencia ###
    if((sum(na.rm=T,diff(valores[1:5])==1)>=4 &
max(na.rm=T,diff(valores[1:5]))==1) | (sum(na.rm=T,diff(valores[1:6])==1)>=4
& max(na.rm=T,diff(valores[1:6]))==1) |
(sum(na.rm=T,diff(valores[1:7])==1)>=4 & max(na.rm=T,diff(valores[1:7]))==1)
| (sum(na.rm=T,diff(valores[2:6])==1)>=4 &
max(na.rm=T,diff(valores[2:6]))==1) | (sum(na.rm=T,diff(valores[2:7])==1)>=4
& max(na.rm=T,diff(valores[2:7]))==1) |
(sum(na.rm=T,diff(valores[3:7])==1)>=4 &
max(na.rm=T,diff(valores[3:7]))==1)) # algum dos possiveis intervalos de
cinco-sete valores tem que ter 4 diffs iguais a 1 entre os valores, e
nenhuma das diffs pode ser maior do que 1 - o criterio de diff so funciona
porque os valores ja sao dados ordenados no argumento
    {
      desseq.temp <- rep(0,6) # criando objeto para ser preenchido abaixo
      if(sum(na.rm=T,diff(valores[1:5])==1)>=4 &
max(na.rm=T,diff(valores[1:5]))==1) # se sequencia estiver entre as posicoes
1 e 5
      {
        desseq.temp[1] <- valores[5] # valor de desempate (maior da
sequencia) esta na posicao 5
      }
      if(sum(na.rm=T,diff(valores[1:6])==1)>=4 &
max(na.rm=T,diff(valores[1:6]))==1) # se entre 1 e 6
      {
        desseq.temp[2] <- valores[6] # na posicao 6
      }
      if(sum(na.rm=T,diff(valores[1:7])==1)>=4 &
max(na.rm=T,diff(valores[1:7]))==1) # se entre 1 e 7
      {
        desseq.temp[3] <- valores[7] # na posicao 7
      }
      if(sum(na.rm=T,diff(valores[2:6])==1)>=4 &
max(na.rm=T,diff(valores[2:6]))==1) # se entre 2 e 6
      {
        desseq.temp[4] <- valores[6] # na posicao 6
      }
      if(sum(na.rm=T,diff(valores[2:7])==1)>=4 &
max(na.rm=T,diff(valores[2:7]))==1) # se entre 2 e 7
      {
        desseq.temp[5] <- valores[7] # na posicao 7
      }
      if(sum(na.rm=T,diff(valores[3:7])==1)>=4 &
max(na.rm=T,diff(valores[3:7]))==1) # se entre 3 e 7
      {
        desseq.temp[6] <- valores[7] # na posicao 7
      }
      desseq <- max(na.rm=T,desseq.temp)  # valor de desempate é o valor
maximo da sequencia
return(c(jogo=5,desemp1=desseq,desemp2=0,desemp3=0,desemp4=0,desemp5=0)) #
```

```
retorna valor do tipo de jogo (5) e do desempate
    }
    ### TRIO = tres cartas de mesmo valor ###
    if(max(table(valores))==3) # algum valor aparece tres vezes?
    {
      destri1 <- as.numeric(names(table(valores)[table(valores)==3])) #
desempate1: valor que aparece tres vezes
      destri2 <- max(setdiff(valores,destri1)) # desempate2: maior carta sem
contar a do trio
      destri3 <- max(setdiff(valores,c(destri1,destri2))) # desempate3:
maior carta sem contar as de cima
return(c(jogo=4,desemp1=destri1,desemp2=destri2,desemp3=destri3,desemp4=0,de
semp5=0)) # retorna valor do tipo de jogo (4) e dos desempates
    }
    ### DOIS PARES = duas cartas de um mesmo valor, duas de outro ###
    if(sort(table(valores),decreasing=T)[1]==2 # o primeiro valor mais
frequente aparece duas vezes?
       & sort(table(valores),decreasing=T)[2]==2) # o segundo valor mais
frequente aparece duas vezes?
    {
      desdp1 <- max(as.numeric(names(table(valores)[table(valores)==2]))) #
1o desempate: maior valor que aparece duas vezes
      desdp2 <- max(as.numeric(names(table(valores)))[table(valores)==2 &
as.numeric(names(table(valores)))!=desdp1]) # 2o desempate: maior valor que
aparece duas vezes, sem contar o 1o
      desdp3 <- max(setdiff(valores,c(desdp1,desdp2))) # 3o desempate: maior
valor que sobrou, compondo a quinta carta
return(c(jogo=3,desemp1=desdp1,desemp2=desdp2,desemp3=desdp3,desemp4=0,desem
p5=0)) # retorna valor do tipo de jogo (3) e dos desempates
    }
    ### PAR = duas cartas de um mesmo valor ###
    if(max(table(valores))==2) # algum valor aparece duas vezes?
    {
      despar1 <- max(as.numeric(names(table(valores)[table(valores)==2]))) #
1o desempate: valor que aparece duas vezes
      despar2 <- max(setdiff(valores,despar1)) # 2o desempate: maior valor
diferente do valor que aparece duas vezes
      despar3 <- max(setdiff(valores,c(despar1,despar2))) # 3o desempate:
maior valor, que não os de cima
      despar4 <- max(setdiff(valores,c(despar1,despar2,despar3))) # 4o
desempate: maior valor, que não os de cima
return(c(jogo=2,desemp1=despar1,desemp2=despar2,desemp3=despar3,desemp4=desp
ar4,desemp5=0)) # retorna valor do jogo (2) e dos desempates
    }
    ### ALTA = ausencia de qualquer dos padrões acima, sendo a carta mais
alta o "jogo" ###
    else
    {
return(c(jogo=1,desemp1=valores[7],desemp2=valores[6],desemp3=valores[5],des
emp4=valores[4],desemp5=valores[3])) # retorna valor do jogo (1) e dos cinco
desempates, que são as cinco maiores cartas da mão, em ordem decrescente
```

```
    }
  }
  jogos.mao <- rep(NA,nsim) # objeto vazio que contera jogos realizados pelo
usuario em cada simulacao
  vencedor <- rep(NA,nsim) # objeto vazio que contera quem venceu cada
simulacao
  #### simulacoes de uma partida completa ####
  for(j in 1:nsim) # simulacao de uma partida completa, repetida nsim vezes
  {
    #### distribuindo cartas ####
    maos <-
matrix(data=rep(x=NA,times=(1+nadv)*7),ncol=1+nadv,nrow=7,dimnames=list(c("m
ao1","mao2","flop1","flop2","flop3","turn","river"))) # matriz vazia que em
que estara a mao de 7 cartas de cada jogador (usuario e adversarios)
    maos[1,1] <- mao1 # primeira carta do usuario foi dada como argumento
    maos[2,1] <- mao2 # segunda carta do usuario tambem foi dada como
argumento
    if(is.null(flop1) == FALSE) # o usuario deu como argumento a carta
flop1?
    {
      maos[3,] <- flop1 # se sim, ela passa a compor a mao de todos os
jogadores (linha 3 de todas as colunas)
    }
    else
    {
      maos[3,] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F) #
se nao, uma carta diferente das que ja estao na matriz "maos" (onde todas as
cartas do jogo - maos e mesa - sao guardadas) eh sorteada
    }
    if(is.null(flop2) == FALSE) # o usuario deu como argumento a carta
flop2?
    {
      maos[4,] <- flop2 # se sim, ela passa a compor a mao de todos os
jogadores (linha 4 de todas as colunas)
    }
    else
    {
      maos[4,] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F) #
se nao, uma carta diferente das que ja estao na matriz "maos" (onde todas as
cartas do jogo - maos e mesa - sao guardadas) eh sorteada
    }
    if(is.null(flop3) == FALSE) # o usuario deu como argumento a carta
flop3?
    {
      maos[5,] <- flop3 # se sim, ela passa a compor a mao de todos os
jogadores (linha 5 de todas as colunas)
    }
    else
    {
      maos[5,] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F) #
```

```
se nao, uma carta diferente das que ja estao na matriz "maos" (onde todas as
cartas do jogo - maos e mesa - sao guardadas) eh sorteada
    }
    if(is.null(turn) == FALSE) # o usuario deu como argumento a carta turn?
    {
      maos[6,] <- turn # se sim, ela passa a compor a mao de todos os
jogadores (linha 6 de todas as colunas)
    }
    else
    {
      maos[6,] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F) #
se nao, uma carta diferente das que ja estao na matriz "maos" (onde todas as
cartas do jogo - maos e mesa - sao guardadas) eh sorteada
    }
    if(is.null(river) == FALSE) # o usuario deu como argumento a carta
river?
    {
      maos[7,] <- river # se sim, ela passa a compor a mao de todos os
jogadores (linha 6 de todas as colunas)
    }
    else
    {
      maos[7,] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F) #
se nao, uma carta diferente das que ja estao na matriz "maos" (onde todas as
cartas do jogo - maos e mesa - sao guardadas) eh sorteada
    }
    for(i in 1:nadv) # as demais cartas serao distribuidas para o numero de
adversarios necessario
    {
      maos[1,1+i] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F)
# sorteio da primeira carta da mao do adversario (setdiff funciona na
iteracao porque conforme cartas vao sendo sorteadas, sao guardadas em "maos"
e portanto excluidas de sorteios subsequentes)
      maos[2,1+i] <- sample(x=setdiff(baralho$carta,maos),size=1,replace=F)
# sorteio da segunda carta
    }
    #### analisando os jogos de cada jogador ####
    numeros.v1 <-
matrix(data=rep(x=NA,times=(1+nadv)*7),ncol=1+nadv,nrow=7,dimnames=list(c("m
ao1","mao2","flop1","flop2","flop3","turn","river"))) # matriz vazia em que
estara a mao de 7 cartas de cada jogador, convertida para seu valor (A
maior)
    numeros.v2 <- numeros.v1 # matriz vazia em que estara a mao de 7 cartas
de cada jogador, convertida para seu valor (A menor)
    numeros.vo1 <- numeros.v1 # matriz vazia em que estara a mao ordenada (A
maior)
    numeros.vo2 <- numeros.v1 # matriz vazia em que estara a mao ordenada (A
menor)
    naipes <- numeros.v1 # matriz vazia em que estarao os naipes
    naipes.o1 <- numeros.v1 # matriz vazia em que estarao os naipes apos a
ordenaca dos valores (A maior)
```

```
    naipes.o2 <- numeros.v1 # matriz vazia em que estarao os naipes apos a
ordenaca dos valores (A menor)
    jogos1 <- rep(NA,6) # vetor vazio em que estara o tipo de jogo obtido
pelo jogador (A maior)
    jogos2 <- jogos1 # vetor vazio em que estara o tipo de jogo obtido pelo
jogador (A menor)
    jogos.df <-
data.frame(jogo=rep(NA,2),desemp1=rep(NA,2),desemp2=rep(NA,2),desemp3=rep(NA
,2),desemp4=rep(NA,2),desemp5=rep(NA,2)) # data frame vazio em que estarao
os dois jogos possiveis (A maior e A menor) de cada jogador ordenados
    jogos.final <-
data.frame(jogador=rep(NA,(1+nadv)),jogo=rep(NA,(1+nadv)),desemp1=rep(NA,(1+
nadv)),desemp2=rep(NA,(1+nadv)),desemp3=rep(NA,(1+nadv)),desemp4=rep(NA,(1+n
adv)),desemp5=rep(NA,(1+nadv))) # data frame em que estarao os maiores
jogos+desempates de todos os jogadores
    for(i in 1:(1+nadv))
    {
      numeros.v1 <- baralho$ordem.num1[match(maos[,i],baralho$carta)] #
convertendo as cartas em seu valor numerico (segundo ordenacao em que A eh
maior)
      numeros.v2 <- baralho$ordem.num2[match(maos[,i],baralho$carta)] #
convertendo as cartas em seu valor numerico (segundo ordenacao em que A eh
menor)
      numeros.vo1 <- sort(numeros.v1) # ordenando numeros.v1
      numeros.vo2 <- sort(numeros.v2) # ordenando numeros.v2
      naipes <- as.character(baralho$naipe[match(maos[,i],baralho$carta)]) #
convertendo as cartas em seu naipe
      naipes.o1 <-
naipes[order(baralho$ordem.num1[match(maos[,i],baralho$carta)])] # ordenando
naipes (A maior) como os numeros.vo1 para que eles fiquem associados (já que
entrarão como argumentos separados na funçao aval.cartas)
      naipes.o2 <-
naipes[order(baralho$ordem.num2[match(maos[,i],baralho$carta)])] # ordenando
naipes (A menor) para ficarem associados aos numeros.vo2
      jogos1 <- aval.cartas(numeros.vo1,naipes.o1) # inserindo valores do
jogo+desempates (resultado da funcao aval.cartas) no vetor
      jogos2 <- aval.cartas(numeros.vo2,naipes.o2) # o mesmo, com A menor
      jogos.df[1,] <- jogos1 # inserindo jogo com A maior no data frame
      jogos.df[2,] <- jogos2 # inserindo jogo com A menor
      jogos.df <-
jogos.df[order(jogos.df$jogo,jogos.df$desemp1,jogos.df$desemp2,jogos.df$dese
mp3,jogos.df$desemp4,jogos.df$desemp5,decreasing=T),] # ordenando o data
frame (descendente) de acordo com os seguintes criterios (em ordem): valor
do tipo de jogo, valor do desempate 1, desempate 2, desempate 3, desempate
4, e desempate 5
      jogos.final[i,] <- c(i,jogos.df[1,]) # inserindo o maior jogo no data
frame (posicao 1 de jogos.df pois a ordenacao era descendente), bem como i
(o numero do jogador; 1 = usuario)
    }
    #### determinando vencedores das partidas simuladas ####
```

```
    jogos.final <-
jogos.final[order(jogos.final$jogo,jogos.final$desemp1,jogos.final$desemp2,j
ogos.final$desemp3,jogos.final$desemp4,jogos.final$desemp5,decreasing=T),] #
ordenando (descendente) o maior jogo de todos os jogadores de acordo com
tipo de jogo, depois desempates
    if(jogos.final$jogador[1]==1 & jogos.final$jogo[1]==jogos.final$jogo[2]
& jogos.final$desemp1[1]==jogos.final$desemp1[2] &
jogos.final$desemp2[1]==jogos.final$desemp2[2] &
jogos.final$desemp3[1]==jogos.final$desemp3[2] &
jogos.final$desemp4[1]==jogos.final$desemp4[2] &
jogos.final$desemp5[1]==jogos.final$desemp5[2]) # se usuario (jogador 1)
estiver na primeira posicao, e seu jogo for exatamente igual ao do segundo
colocado (incluindo todos os desempates)... (estando as outras colunas
identicas o usuario sempre estara na primeira posicao pois foi o primeiro
valor inserido no data frame (i=1 no for) e a coluna "jogador" nao foi
ordenada)
    {
      vencedor[j] <- 0 # ... atribui valor 0 no vetor vencedor (empate)
    }
    else
    {
    vencedor[j] <- jogos.final$jogador[1] # ... se não, vencedor esta na
primeira linha do df jogos.final - apenas empates envolvendo o usuario
interessam, caso contrario eh o mesmo que derrota (ou seja, empates entre
dois outros adversarios serao incluidos aqui e nao em empate)
    }
    jogos.mao[j] <- jogos.final[jogos.final$jogador==1,2] # atribuindo os
jogos (2a coluna) que o usuario (jogador=1) fez ao objeto jogos.mao
  }
  #### output ####
  vencer <- round(sum(vencedor==1)/nsim*100,digits=2) # % de jogos vencidos
  empatar <- round(sum(vencedor==0)/nsim*100,digits=2) # % de jogos
empatados
  perder <- round(sum(vencedor>1)/nsim*100,digits=2) # % de jogos perdidos
  # abaixo: data frame com a % em que cada jogo apareceu na mao do usuario
  chances.jogos <-
data.frame("nenhum"=round(sum(jogos.mao==1)/nsim*100,digits=2), # numero de
vezes que o valor 1 (é como a funcao aval.cartas retorna ausencia de jogos)
aparece em jogos.mao, divido por nsim e multiplicado por 100
"par"=round(sum(jogos.mao==2)/nsim*100,digits=2), # idem para o valor 2
(par)
                              "dois
pares"=round(sum(jogos.mao==3)/nsim*100,digits=2), # idem para dois pares
"trio"=round(sum(jogos.mao==4)/nsim*100,digits=2), # idem para trio
"sequência"=round(sum(jogos.mao==5)/nsim*100,digits=2), # idem para
sequencia
"flush"=round(sum(jogos.mao==6)/nsim*100,digits=2), # idem para flush
                              "full
house"=round(sum(jogos.mao==7)/nsim*100,digits=2), # idem para full house
"quadra"=round(sum(jogos.mao==8)/nsim*100,digits=2), # idem para quadra
                              "straight
```

```
flush"=round(sum(jogos.mao==9)/nsim*100,digits=2)) # idem para straight
flush
  rownames(chances.jogos) <- "" # excluindo nomes de linhas automaticamente
atribuidos
  # abaixo: data frame com a % que cada jogo ou um jogo superior apareceu na
mao do usuario
  chances.acum <-
data.frame("nenhum"=round(sum(jogos.mao>=1)/nsim*100,digits=2),
"par"=round(sum(jogos.mao>=2)/nsim*100,digits=2), "dois
pares"=round(sum(jogos.mao>=3)/nsim*100,digits=2),
"trio"=round(sum(jogos.mao>=4)/nsim*100,digits=2),
"sequência"=round(sum(jogos.mao>=5)/nsim*100,digits=2),
"flush"=round(sum(jogos.mao>=6)/nsim*100,digits=2), "full
house"=round(sum(jogos.mao>=7)/nsim*100,digits=2),
"quadra"=round(sum(jogos.mao>=8)/nsim*100,digits=2), "straight
flush"=round(sum(jogos.mao>=9)/nsim*100,digits=2)) # analogo ao data frame
anterior, mas sempre >= ao valor do jogo (ao inves de ==) pois queremos
saber a chance de conseguir aquele jogo ou melhor
  rownames(chances.acum) <- "" # excluindo nomes de linhas
  return(list("Chance (%) de cada resultado" =
c("vitória"=vencer,"empate"=empatar,"derrota"=perder),"Chance (%) de
realizar cada tipo de jogo" = chances.jogos,"Chance (%) de realizar cada
tipo de jogo ou melhor" = chances.acum)) # retorna lista com: chance de cada
resultado, chance de realizar cada jogo, e chance de cada jogo ou melhor
}
```

Página de ajuda da função

Arquivo da página de ajuda: help_prob.poker.txt

```
prob.poker                package:unknown                R Documentation

Chances de vencer uma partida de pôquer

Description:

    A função faz simulações de partidas de pôquer Texas Hold'Em para estimar
as
chances do usuário vencer, empatar, ou perder a partida, bem como as chances
de realizar
cada tipo de jogo, com as cartas que tem na mão.

Usage:

    prob.poker(mao1, mao2, flop1 = NULL, flop2 = NULL, flop3 = NULL, turn =
NULL,
river = NULL, nadv, nsim = 1000)
```

```
Arguments:

    mao1,mao2        Caracteres contendo valor e naipe das cartas que estão
na
                mão do usuário. Veja seção Note para o formato correto de
                argumentos de carta.

    flop1,flop2,flop3    Caracteres contendo valor e naipe da primeira,
segunda e
                terceira cartas (flop) que foram abertas sobre a mesa. Se
                NULL (padrão), cartas serão sorteadas. Veja Note para o
                formato correto de argumentos de carta.

    turn            Caractere contendo valor e naipe da quarta carta (turn)
                que foi aberta sobre a mesa. Se NULL (padrão), carta será
                sorteada. Veja Note para o formato correto de argumentos
                de carta.

    river            Caractere contendo valor e naipe da quinta carta
(river)
                que foi aberta sobre a mesa. Se NULL (padrão), carta será
                sorteada. Veja Note para o formato correto de argumentos
                de carta.

    nadv            Número de adversários contra os quais o usuário está
                jogando.

    nsim            Número de partidas a serem simuladas, padronizado para
                1000.

Details:

    Quatro vezes em uma partida de pôquer Texas Hold’em, o jogador precisa
tomar uma
importante decisão: largar a mão, pagar a aposta, ou aumentá-la. Essa
decisão parte de
uma pergunta importante: "quais as chances de ser vencedor ao final da
partida, com as
cartas que tenho?". Essa função pode ser usada em cada uma das rodadas de
aposta (pré
flop, flop, turn e river) para embasar a decisão do jogador e permitir que
ele aposte
(ou não) de acordo com as perspectivas reais de vitória.
    Em cada partida simulada, a função sorteia as cartas necessárias (i.e.,
as
cartas das mãos dos adversários e, se não foram inseridas como argumentos,
as que serão
abertas sobre a mesa), avalia o melhor jogo final de cada adversário, e
determina o
vencedor. As chances de cada resultado para o usuário (vitória, empate,
derrota) são
```

```
obtidas a partir dos vencedores das simulações. As chances de cada tipo de
jogo são
obtidas a partir dos jogos obtido pelo usuário nas simulações.

Value:

    A função retorna um objeto da classe list com os seguintes componentes:

    Chance (%) de cada    porcentagem das partidas simuladas que foram
    resultado        vencidas pelo usuário (vitória), vencidas pelo usuário
                e um adversário empatados (empate), e vencidas por
                adversários (derrota).
    Chance (%) de realizar    porcentagem das partidas simuladas ao
    cada tipo de jogo    final das quais o usuário obteve cada tipo de jogo:
                nenhum, par, dois pares, trio, sequência, flush, full
                house, quadra, straight flush.

    Chance (%) de realizar    porcentagem das partidas simuladas ao final
das quais o
    cada tipo de jogo ou    usuário obteve cada tipo de jogo ou um jogo
superior.
    melhor

Warning:

    A função é interrompida em três situações:

    Quando qualquer carta inserida como argumento não pertence ao baralho
virtual de
    52 cartas, no formato apropriado (ver Note).

    Quando uma mesma carta é dada como argumento mais de uma vez.

    Quando o número de adversários é superior a 22, ou seja, quando não há
cartas
    suficientes para serem sorteadas para todos os adversários

Note:

    Todos os argumentos de cartas precisam estar no seguinte formato: "valor
da
carta", "primeira letra do naipe" (e.g., 10 de copas é "10c", rei de paus é
"Kp"). Os 52
valores possíveis de serem inseridos como argumentos de carta se encontram
no vetor
baralho.simp, em Examples. A função diferencia maiúsculas de minúsculas,
portanto letras
de valores (J, Q, K, A) tem que estar em caixa alta e letras de naipe (c, o,
p, e), em
caixa baixa. A ordem em que as cartas da mão (mao1, mao2) e da mesa (flop1,
```

```
flop2, flop3,
turn, river) são inseridas não é importante.
    Um maior número de simulações (nsim) faz com que as chances retornadas
pela
função sejam mais precisas, porém também faz com que o tempo de
processamento seja
maior. Quanto mais argumentos de carta são dados, mais precisas são as
chances
calculadas, pois há menos cartas para serem simuladas.
    A função tem certas limitações, como não considerar as possíveis
decisões de
jogadores nas apostas subsequentes. Assim, por exemplo, o número de
adversários inseridos como argumento se manterá até o fim das partidas
simuladas, sem
considerar que eles poderiam sair em uma das futuras rodadas de aposta.
Consequentemente, as chances de vencer do usuário serão mais baixas quanto
maior o nadv,
pois aumenta a probabilidade de qualquer adversário ter um jogo superior ao
do usuário.

Author(s):

    João C. T. Menezes
    jocateme@gmail.com

Examples:

### Criando baralho virtual e sorteando cartas que serão utilizadas
baralho.simp <- paste(c(rep(2:10, each = 4), rep(c("J", "Q", "K", "A"), each
= 4)),
rep(c("p", "c", "o", "e"), times = 13), sep = "")
sample <- sample(baralho.simp,size=7) ## sorteando sete cartas para serem
utilizadas
no exemplo
mao1 <- sample[1]
mao2 <- sample[2]
flop1 <- sample[3]
flop2 <- sample[4]
flop3 <- sample[5]
turn <- sample[6]
river <- sample[7]

### Exemplo do avanço de uma mesma partida heads-up (i.e., usuário contra um
adversário)
### chances pré-flop
prob.poker(mao1, mao2, nadv = 1)

### chances após flop
prob.poker(mao1, mao2, flop1, flop2, flop3, nadv = 1)

### chances após turn
```

```
prob.poker(mao1, mao2, flop1, flop2, flop3, turn, nadv = 1)

### chances após river
prob.poker(mao1, mao2, flop1, flop2, flop3, turn, river, nadv = 1)
```