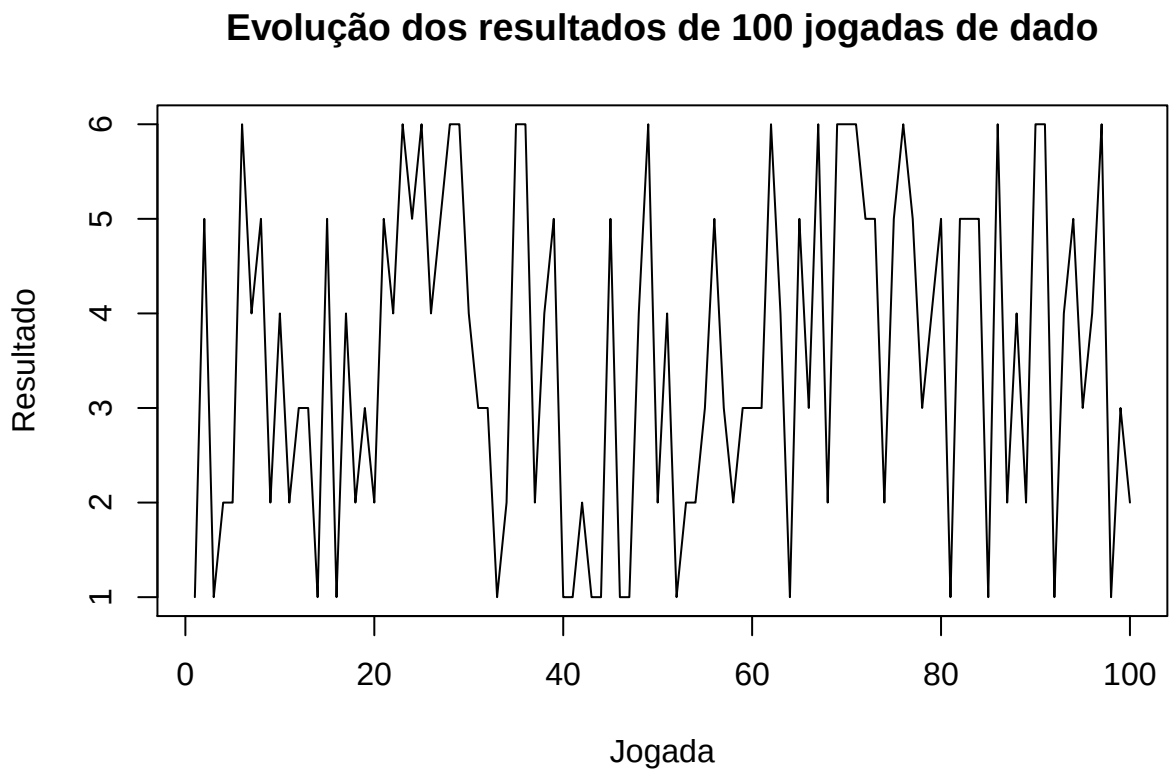
Uma introdução prática ao Modelo Markov de gramática

João Paulo Lazzarini Cyrino

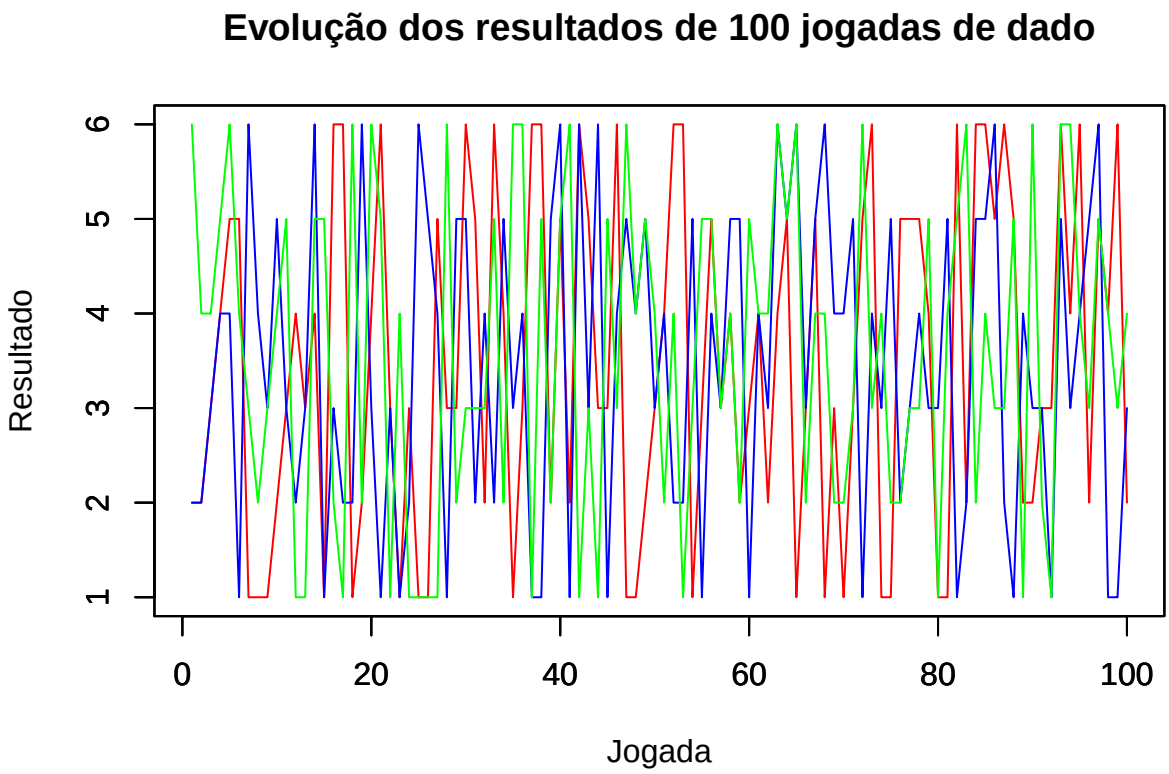
26/09/2020

Processo Estocástico

Um **processo estocástico** (aleatório) pode representar a evolução de valores de uma variável aleatória ao longo do tempo. Por exemplo, vamos jogar um dado 100 vezes e anotar cada resultado. Podemos representar os valores da variável resultado do dado ao longo de cada jogada com o gráfico abaixo:



Esses resultados, no entanto, serão diferentes a cada vez que repetirmos o experimento. Abaixo mostramos um gráfico com 3 repetições do experimento, cada repetição é representada por uma cor (vermelho, verde e azul):



No exemplo do lançamento dos dados, o resultado de cada jogada é independente um do outro. Dessa forma, podemos ter um número muito grande de possibilidades de como esses resultados ocorrem em cada jogada, mais precisamente, 6^{100}.

Sucede que alguns processos estocásticos podem ter alguma memória de eventos anteriores. Há, especificamente um que se desenvolve a partir da memória do evento imediatamente anterior, que é a *Cadeia de Markov*. Vamos explicar a esse tipo de processo utilizando a própria língua como um exemplo.

A Língua como um Processo Estocástico

Sob um ponto de vista muito simples, podemos considerar que a parte observável de uma língua são sequências de símbolos (fonemas, morfemas, sinais, caracteres, enfim). Damos a essas sequencias nomes como frases, textos, etc.

Muito bem. Suponhamos que tivessemos um dado de várias faces, cada face representando um fonema do português, ou, para simplificar, um caractere de a-z mais o caractere de espaço. Se lançássemos esse dado 100 vezes e anotássemos o resultado, seria alta a probabilidade de obtermos frases que se assemelham a frases escritas do português?

Podemos fazer esse experimento na linguagem R. Vamos criar uma lista de caracteres que nos interessam. Concretamente, a função `letters` já nos dá uma lista de caracteres de a-z. Vamos adicioná-la à uma lista, juntamente com os caracteres , `.` e`,`, representando espaço, ponto e vírgula, respectivamente.

```
# Criar lista de caracteres:
c <- c(letters, ' ', '.', ',')
```

Agora, podemos utilizar a função `sample` para criar uma lista de 100 caracteres sorteados a partir de nossa lista `c`. Utilizamos a função `paste` para juntar tudo em uma única string:

```
lista <- sample(c,100,replace=TRUE)
paste(lista, collapse='')
```

```
## [1] "tmjrvnnz, lmwnnluwm,vyddf zjxaa njdydxulmwogcgsvzzapkhvcva gt scwskxbg zuchdzxqnwen,uuxrvrpn,gyl
```

Obtivemos um texto inteligível? Provavelmente não!

Você, linguista, deve estar pensando: é óbvio que não, afinal de contas, em uma língua a distribuição desses símbolos segue algum tipo de regra. E, sim, é fato que há algo de regular nessas sequências de símbolos. Normalmente, se quisermos gerar uma sequência de símbolos inteligível para um falante de português, um linguista será bastante inclinado a dizer que isso só pode ser feito sujeitando nossos símbolos a regras estabelecidas a priori (regras fonológicas, para combinações de fonemas e regras morfossintáticas, para combinação de morfemas/palavras).

Acontece que é possível gerar textos inteligíveis em português com métodos aleatórios. O que falta para nosso experimento começar a produzir é que, em nosso processo estocástico, esteja envolvida a memória, ao menos, do evento anterior. Isso é: a probabilidade de ocorrer um caractere depende do caractere imediatamente anterior a ele.

Vamos ilustrar isso da seguinte forma, na variável `corpus` armazenamos um texto em português, concretamente um conto de Grimm.

Com a função `strsplit` criamos um vetor dos caracteres que temos em `corpus`. OBS: `strsplit` nos retorna uma lista de vetores, então precisamos acessar o primeiro vetor com o operador [[1]].

```
# Vetor de Caracteres a partir do corpus
car <- strsplit(corpus, split="")[[1]]
```

O vetor `car` contém todos os caracteres do corpus em sequência. Queremos fazer um algoritmo que monte um texto da seguinte forma:

1. Escolher um caractere aleatoriamente
2. Anotar o caractere imediatamente posterior em uma string
3. Escolher um aleatoriamente um caractere idêntico a esse anotado
4. Repetir a partir de 2.

Para fazer isso em R de maneira eficiente faremos uma tabela que associe cada caractere ao vizinho imediato. Uma das colunas terá do primeiro ao penúltimo caractere do corpus e a outra do segundo ao último. Veja como isso pode ser feito:

```
# Criar a matriz e converter em data.frame
tabela <- data.frame(cbind(car[1:length(car)-1],car[2:length(car)]))
# Dar nomes 'i' e 'j' às colunas:
colnames(tabela) <- c('i','j')
# Mostrar primeiras 10 linhas:
tabela[1:10,]
```

```
##    i j
## 1  H o
## 2  o u
## 3  u v
## 4  v e
## 5  e ,
## 6  ,
## 7    u
## 8  u m
## 9  m a
## 10 a
```

Agora, para o algoritmo:

```
gerar.texto <- function(n) {
  # Escolher aleatoriamente uma linha da tabela:
  l <- sample(c(1:nrow(tabela)),1)
  # Capturar o elemento e seguinte:
  e <- tabela$j[l]
  # Criar string
  s <- ""
  # Repetir n vezes
  for (i in 1:n) {
    # Anotar na string o elemento e:
    s <- paste(s,e,sep="")
    # Escolher, aleatoriamente uma linha que tenha o elemento e
    # na primeira coluna e capturar o elemento seguinte
    e <- sample(tabela[tabela$i==e,2],1)
  }
  # Retornar a string
  s
}
# Chamar a função com 100 caracteres
gerar.texto(100)
```

```
## [1] "oinde stimefendonua dicairaza fossco disfla ecosse tam corastr, co strmo meu. ento lou teu caca
```

Você acha que o resultado ficou *ligeiramente* mais interessante/semelhante ao português? Perceba que essa semelhança vai se dar em um nível ortográfico apenas, já que estamos lidando com a distribuição dos caracteres de um texto de língua portuguesa. Podemos melhorar a semelhança aumentando a memória para, ao invés do último caractere, os dois últimos, por exemplo. Mas especificamente o *processo estocástico* que envolve apenas a memória do úlitmo evento é denominado *Cadeia de Markov* e o que acabamos de fazer é um gerador de sequências de caracteres baseado em uma *Cadeia de Markov*.

Esse tipo de gerador torna-se surpreendente quando, ao invés de sequências de caracteres, começamos a utilizar sequências de palavras. Veremos isso adiante.

Modelo Markov de Gramática

Vamos retomar nosso corpus e, ao invés de extrair uma sequência de caracteres, vamos extrair as palavras. Para fazer isso, precisamos criar um algoritmo que separe as palavras com a função `strsplit`. Também vamos usar as funções `gsub` e `setdiff` para fazer algumas limpezas no corpus. Abaixo temos uma forma (não das mais elegantes) de obter um vetor de palavras:

```
# Substituir sinais de quebra de linha por " "
corpus.novo <- gsub("\n"," ",corpus)
# Substituir sinais de pontuação por ""
corpus.novo <- gsub("[.,;:!?]","",corpus.novo)
# Obter o vetor de palavras
pal <- strsplit(corpus.novo," ")[[1]]
# Excluir strings vazias do vetor
pal <- setdiff(pal, "")
# 100 primeiras palavras
pal[1:100]
```

```
##   [1] "Houve"        "uma"          "vez"          "um"           "camponês"
##   [6] "que"          "tinha"        "filho"        "do"           "tamanho"
##  [11] "de"           "polegar"      "mas"          "ao"           "chegar"
##  [16] "à"            "adolescência" "não"          "crescido"     "nem"
```

```
##  [21] "linha"        "mais"         "Certa"        "em"           "o"
##  [26] "se"           "dispunha"     "a"            "sair"         "para"
##  [31] "campo"        "e"            "arar"         "terra"        "pimpolho"
##  [36] "chegou-se"    "ele"          "disse"        "-"            "Pai"
##  [41] "leva-me"      "contigo"      "Queres"       "ir"           "perguntou"
##  [46] "pai"          "é"            "melhor"       "ficares"      "aqui"
##  [51] "lá"           "ajudas"       "nada"         "além"         "disso"
##  [56] "poderias"     "perder-te"    "Polegarzinho" "então"        "pôs-se"
##  [61] "chorar"       "Para"         "amolasse"     "meteu-o"      "no"
##  [66] "bolso"        "levou-o"      "consigo"      "Chegando"     "tirou"
##  [71] "pequeno"      "acomodou-o"   "num"          "sulco"        "recém-aberto"
##  [76] "deixando-o"   "sentado"      "nisso"        "veio"         "descendo"
##  [81] "da"           "montanha"     "enorme"       "gigante"      "apontando-o"
##  [86] "menino"       "pondo-lhe"    "medo"         "ficasse"      "quietinho"
##  [91] "Estás"        "vendo"        "aquele"       "monstro"      "Ele"
##  [96] "vem"          "buscar-te"    "Com"          "as"           "longas"
```

Vamos agora criar nossa tabela de palavras bigramas (palavra e sua palavra vizinha):

```
tabela <- data.frame(cbind(pal[1:length(pal)-1],pal[2:length(pal)]))
colnames(tabela) <- c("i","j")
tabela[1:10,]
```

```
##           i        j
## 1     Houve      uma
## 2       uma      vez
## 3       vez       um
## 4        um camponês
## 5  camponês      que
## 6       que    tinha
## 7     tinha    filho
## 8     filho       do
## 9        do  tamanho
## 10  tamanho       de
```

Agora vamos recriar a função que gera um texto a partir de nossa tabela:

```
gerar.texto <- function(n) {
  # Escolher aleatoriamente uma linha da tabela:
  l <- sample(c(1:nrow(tabela)),1)
  # Capturar o elemento e seguinte:
  e <- tabela$j[l]
  # Criar string
  s <- ""
  # Repetir n vezes
  for (i in 1:n) {
    # Anotar na string o elemento e:
    s <- paste(s,e,sep=" ")
    # Escolher, aleatoriamente uma linha que tenha o elemento e
    # na primeira coluna e capturar o elemento seguinte
    e <- sample(tabela[tabela$i==e,2],1)
  }
  # Retornar a string
  s
}
# Chamar a função com 100 caracteres
```

```
gerar.texto(100)
```

```
## [1] " o se dispunha a sair para campo e arar terra pimpolho chegou-se ele disse - Pai leva-me contig
```

Os resultados são um tanto quanto impressionantes, não?

O que acabamos de fazer foi um gerador de textos a partir de uma gramática Markov do português. Também chamamos isso de modelo de bigramas, ou melhor, de n-gramas, já que pode haver modelos que levem em conta uma memória maior do que a da palavra imediatamente anterior.

Obviamente, há muito o que fazer (especialmente com relação à pontuação) para tornar os textos gerados por esse tipo de modelo mais parecidos com um texto de língua portuguesa. Mas já é um ponto de partida para entendermos que existe um caminho de se entender a gramática das línguas sob uma perspectiva probabilística.