Análise textual e previsão 1

Baseado em: “*Notes on Naive Bayes Classifiers for Spam Filtering* ”

Por: Pedro de Araújo Ribeiro

Considerações:

- Este não é um estudo compreensivo de RegEx
- Assume-se conhecimentos básicos da linguagem R de programação
- Utilizamos os dados gerados na apresentação anterior
- Bibliotecas usadas:
  - tidyverse
  - dplyr
- Github
- Kaggle

Proposta:

Tendo feito a extração de dados com web scraping, aplicaremos técnicas estatísticas e de análise de texto para realizar classificações de notícias.

As técnicas e metodologias utilizadas aqui são aplicáveis a diversos problemas de classificação como, por exemplo, filtros de spam.

Inicialmente nossa variável resposta será o tema da notícia e a variável de estudo será somente o título.

Relembrando o formato do conjunto:

Página de download do conjunto

```r
news <- read.csv("News3.csv",header=T,sep=',')
```

```
news                    3928 obs. of 7 variables
  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
  $ Title       : chr  "Candy factory didn't evacuate concerned workers before Pennsylv…
  $ Description: chr  "An eastern Pennsylvania candy factory didn’t evacuate its emplo…
  $ Body        : chr  "An eastern Pennsylvania candy factory didn’t evacuate its emplo…
  $ Keywords    : chr  "accident investigations, accidents, accidents, disasters and sa…
  $ Theme       : chr  "us" "us" "us" "us" ...
  $ Link        : chr  "https://edition.cnn.com/2023/10/06/us/pennsylvania-candy-factor…
```

Preparações: separando os títulos

Criamos um vetor separado para os títulos.

```
titulos <- news$Title
```

Em seguida aplicamos a função `separate_longer_delim` que recebe um data frame, um vetor proporcional as linhas do data frame e um caractere que será usado para separar a string em partes menores.

```
words<-separate_longer_delim(tibble(titulos),titulos,delim=' ')
```

O objeto words será um data frame com uma única variável “titulos” onde cada linha é uma palavra.

Preparações: separando os títulos em palavras

Neste momento ainda estamos apenas construindo um dicionário com todas as palavras do conjunto, portanto aplicaremos a função unique para eliminar repetições.

```
words <- unique(words)
```

Em seguida podemos inspecionar nosso objeto words para termos uma ideia dos tipos de palavras que foram extraídas e como estão escritas.

Preparações: separando os títulos em palavras

Ao fazer isso, notamos que a função incluiu o caractere de fim de linha. Podemos removê-lo com a linha de código abaixo.

```
words <- words[-15,1]
```

Vale notar que a posição será diferente dependendo do conjunto então é sempre recomendável verificar manualmente.

Além disso, uma verificação mais extensa das palavras mostra que temos palavras iguais porém com letra maiúscula, vírgulas, números, pontos e aspas. Todos são diferenças insignificantes semanticamente mas que levam a existência de palavras diferentes no dicionário.

	titulos
5	concerned
6	workers
7	before
8	Pennsylvania
9	explosion
10	that
11	killed
12	7,
13	OSHA
14	finds
15	
16	Baltimore
17	police

Tratamento: maíusculas e pontos

Nossa primeira função de tratamento será `tolower(palavra)` que recebe uma string e a retorna em lowercase.

Em seguida usaremos `gsub('[,.?:]','',palavra)` que recebe um tipo de substring (ou seja, componente de uma string) a ser substituído, a string que entrará no lugar e a string original.

O componente `[,.?:]` da chamada da função está em regex e se refere a toda e qualquer instância dos caracteres entre `[]`, que será substituída por uma string vazia `''`. Ao final teremos:

```
palavra <- tolower(palavra)

palavra <- gsub('[,.?:]','',palavra)
```

Tratamento: aspas simples

Nos títulos não foi observado o uso de aspas duplas, portanto realizaremos apenas o tratamento das simples. Para isso usaremos a função `gsub('\'','',palavra)` porém nem toda instância de aspas simples deve ser removida, palavras como don’t ou Denny’s devem continuar enquanto ‘war crimes’ deve ser transformado em war crimes.

Portanto utilizaremos `str_detect(palavra,".\'.")` para ver se a palavra começa ou termina com aspas simples ou se elas são parte da palavra de fato.

Em regex o ponto `.` indica qualquer caractere (menos vazio ou fim de linha) enquanto o `\'` representa a aspas simples uma vez que `'` sozinha possui significado especial para as linguagens de programação e devemos utilizar a barra `\` para transformá-la em um caractere normal.

Tratamento: aspas simples

Desta forma a função `str_detect(palavra,".\'.")` detecta todas palavras com aspas simples que devem mantê-las, então criaremos um condicional para remover as aspas das demais.

Teremos:

```
if(!(str_detect(palavra,".\'.")))
    palavra <- gsub('\'','',palavra)
```

Tratamento: idades

Palavras diferentes podem vir a significar a mesma coisa em termos de função dentro de uma frase, nos títulos podemos observar diferentes instancias de número-year-old ou número-years, onde o número de fato não importa e apenas o fato de que estamos nos referindo a uma idade.

Portanto utilizaremos `str_detect(palavra,".-year?")` onde o `.` significa qualquer caractere menos o vazio e o fim de linha e o `?` indica qualquer caractere sem restrição.

Teremos:

```
if(str_detect(palavra,".-year?"))
    palavra <- 'II_idade_II'
```

Tratamento: dinheiro

A mesma lógica se aplica para menções de dinheiro independente do valor, aqui usaremos `str_detect(palavra,"\\$.")` onde o `.` significa o mesmo que antes e `\\$` significa o caractere `$`. Uma vez que este possui significado especial devemos utilizar a barra, porém `\$` também não é aceita pelo R então fazemos `\\$`.

Teremos:

```
if(str_detect(palavra,"\\$."))
    palavra <- 'II_dinheiro_II'
```

Tratamento: números

A mesma lógica se aplica para menções de números quaisquer que sejam, aqui usaremos `str_detect(palavra,"^[0-9]*$")` onde a combinação `^ $` significa que o componente pode estar em qualquer lugar entre o início e fim da linha, `[0-9]` significa qualquer dígito e `*` significa que o componente anterior, neste caso `[0-9]`, pode se repetir inúmeras vezes.

Teremos:

```
if(str_detect(palavra,"^[0-9]*$"))
    palavra <- 'II_numero_II'
```

Tratamento: artigos, conectivos, pronomes e afins

Há uma quantidade e variedade imensa de palavras curtas que se repetem várias vezes sem significado maior como the, he/she, in, an, e assim por diante. Sendo assim a função `str_length(palavra)<=3` irá detectar a maioria desses casos de palavras “insignificantes”.

Teremos:

```
if(str_length(palavra)<=3)
    palavra <- 'II_curta_II'
```

Tratamento: função de tratamento

Agora juntaremos todos os condicionais estabelecidos em uma única função:

```
tratamento <- function(palavra){
  palavra <- tolower(palavra)
  palavra <- gsub('[,.?:]','',palavra)
  if(!(str_detect(palavra,".\'."))){palavra <- gsub('\'','',palavra)}
  if(str_detect(palavra,".-year?")){palavra <- 'II_idade_II'}
  if(str_detect(palavra,"\\$.")){palavra <- 'II_dinheiro_II'}
  if(str_detect(palavra,"^[0-9]*$")){palavra <- 'II_numero_II'}
  if(str_length(palavra)<=3){palavra <- 'II_curta_II'}
  return(palavra)
}
```

Montando o dicionário:

Agora resta apenas aplicar o tratamento em todas as palavras extraídas.

```
for(i in 1:nrow(words)){
  words$titulos[i]<-tratamento(words$titulos[i])
}
```

Ao final teremos algumas repetições, então aplicaremos a função abaixo para resolvê-las:

```
dicionario <- distinct(words)
```

Construindo a matriz de repetições:

Nosso próximo objetivo será construir uma matriz que expressa a relação entre frequência de palavras e notícia, começaremos com uma matriz de zeros e depois realizaremos o povoamento:

```
zeros <- matrix(0,nrow=length(titulos),ncol=nrow(dicionario)+2)
```

Aqui as colunas serão as palavras e as linhas as notícias. Depois transformamos esse objeto em um data frame:

```
coords <- data.frame(row.names = titulos,zeros)
```

E as colunas serão nomeadas de acordo com as palavras:

```
colnames(coords)<-c("II_noticia_II","II_tema_II",dicionario$
titulos)
```

Construindo a matriz de repetições:

Antes do povoamento, podemos remover a matriz zeros criada anteriormente com:

```
rm(zeros)
```

E também preencher as colunas da notícia e do tema:

```
coords$II_noticia_II <- titulos
```

```
coords$II_tema_II <- news$Theme
```

Povoando a matriz de repetições:

A função abaixo realizará o povoamento, entraremos em detalhe sobre cada linha em seguida:

```
for(i in 1:length(titulos)){
  titulo <- c(titulos[i])
  words <- str_split_1(titulo," ")
  words <- head(words,-1)
  for(ii in 1:length(words)){
    a<- which(dicionario$titulos==tratamento(words[ii]))
    coords[i,a+2] <- (coords[i,a+2] +1)
  }
}
```

Povoando a matriz de repetições:

Divide uma string em um vetor de strings baseado em outra substring:

```
words <- str_split_1(titulo," ")
```

Remove o fim de linha do vetor:

```
words <- head(words,-1)
```

Identifica a posição no vetor dicionário referente a palavra em questão:

```
a<- which(dicionario$titulos==tratamento(words[ii]))
```

Utilizando a posição anterior, incrementa em 1 a respectiva palavra no dataframe:

```
coords[i,a+2] <- (coords[i,a+2] +1)
```

Matriz de temas e palavras:

Primeiro determinamos os temas possíveis:

```
temas <- unique(coords$II_tema_II)
```

Em seguida criamos outra matriz de zeros como fizemos anteriormente:

```
zeros2 <- matrix(0,nrow=length(temas),ncol=nrow(dicionario)+1)
```

A transformamos em data frame e alteramos os nomes:

```
coords_t <- data.frame(row.names = temas,zeros2)
colnames(coords_t)<-c("II_tema_II",dicionario$titulos)
```

Matriz de temas e palavras:

Preenchemos a coluna dos nomes:

```
rm(zeros2)
```

E por fim removemos a matriz de zeros:

```
coords_t$II_tema_II<-temas
```

Agora resta povoar a nova matriz, explicaremos as linhas em seguida:

```
for(i in 1:length(temas)){
  temp <- coords[coords[,2]==temas[i],]
  for(ii in 2:ncol(coords_t)){
    coords_t[i,ii] <- sum(temp[,ii+1])}}
```

Matriz de temas e palavras:

Cria um data frame temporário apenas com as linhas que tem um tema específico:

```
temp <- coords[coords[,2]==temas[i],]
```

Substitui a posição da matriz associada a essa palavra e notícia pela soma de sua frequência:

```
coords_t[i,ii] <- sum(temp[,ii+1])
```

No fim podemos remover o data frame temporário:

```
rm(temp)
```

Preparação para previsão:

O método que iremos utilizar inicialmente, Naive Bayes, necessitará a inversão das colunas pelas linhas e também garantir que os números da matriz são reconhecidos como números pelo R.

Inverte a matriz e remove a coluna com os temas em formato de string:

```
coords_t <- data.frame(t(coords_t))[-1,]
```

Transforma em número:

```
coords_t<-mutate_all(coords_t,function(x) as.numeric(as.character(x)))
```

Preparação para previsão: treino e teste

Para trabalhar com modelos de previsão é de extrema importância a separação do conjunto original em subconjuntos de treino e teste, portanto voltaremos para antes da criação da matriz de temas e palavras e realizaremos a separação:

```
ran <- sample(1:length(titulos), 0.8 * length(titulos))

treino <- coords[ran,]

teste <- coords[-ran,]
```

Além disso, todas as instâncias seguintes de `coords` devem ser substituídas por treino.

Preparação para previsão: treino e teste

```
temas <- unique(treino$II_tema_II)
zeros2 <- matrix(0,nrow=length(temas),ncol=nrow(dicionario)+1)
coords_t <- data.frame(row.names = temas,zeros2)
colnames(coords_t)<-c("II_tema_II",dicionario$titulos)
rm(zeros2)
coords_t$II_tema_II<-temas
for(i in 1:length(temas)){
  temp <- treino[treino[,2]==temas[i],]
  for(ii in 2:ncol(coords_t)){
    coords_t[i,ii] <- sum(temp[,ii+1])}}
```

Naive Bayes: teoria

Explicaremos Naive Bayes utilizando um exemplo binário onde tentaremos prever se um email é spam ou não baseado nas palavras que contêm. Para isso representaremos um email como uma sequência de palavras {x1,x2,...,xN}.

Nosso objetivo inicial será calcular a probabilidade de spam dado que possui uma palavra específica V, o que representaremos como P(S|V). Pelo teorema de Bayes teremos:

$$\mathbb{P}(S \mid V) = \frac{\mathbb{P}(V \mid S)\mathbb{P}(S)}{\mathbb{P}(V \mid S)\mathbb{P}(S) + \mathbb{P}(V \mid \overline{S})\mathbb{P}(\overline{S})}$$

Para relembrar:

P(S) é a quantidade de spams sobre o total

P(V|S) é a quantidade de spams com a palavra V sobre o total de spams

P($\overline{S}$) é o complemento de S, vulgo os não spams sobre o total

P(V|$\overline{S}$) é a quantidade de não spams com a palavra V sobre o total de não spams

Naive Bayes: teoria

Expandindo a equação anterior para incluir a sequência de palavras teremos ($\overline{S}$ foi substituído por H):

$$\frac{\mathbb{P}(x_1, \ldots, x_n \mid S)\mathbb{P}(S)}{\mathbb{P}(x_1, \ldots, x_n)} = \frac{\mathbb{P}(x_1, \ldots, x_n \mid S)\mathbb{P}(S)}{\mathbb{P}(x_1, \ldots, x_n \mid S)\mathbb{P}(S) + \mathbb{P}(x_1, \ldots, x_n \mid H)\mathbb{P}(H)}$$

E pela definição de probabilidade condicional podemos fazer a seguinte substituição:

$$\mathbb{P}(x_1, \ldots, x_n \mid S)\mathbb{P}(S) = \mathbb{P}(x_1, \ldots, x_n, S)$$

Com isso agora podemos decompor a probabilidade maior em várias sequências de condicionais:

$$\begin{aligned}
\mathbb{P}(x_1, \ldots, x_n, S) &= \mathbb{P}(x_1 \mid x_2, \ldots, x_n, S)\mathbb{P}(x_2, \ldots, x_n, S) \\
&= \mathbb{P}(x_1 \mid x_2, \ldots, x_n, S)\mathbb{P}(x_2 \mid x_3, \ldots, x_n, S)\mathbb{P}(x_3, \ldots, x_n, S) \\
&= \ldots \\
&= \mathbb{P}(x_1 \mid x_2, \ldots, x_n, S)\mathbb{P}(x_2 \mid x_3, \ldots, x_n, S) \ldots \mathbb{P}(x_{n-1} \mid x_n, S)\mathbb{P}(x_n \mid S)\mathbb{P}(S)
\end{aligned}$$

Naive Bayes: teoria

Contudo, todas essas transformações não nos ajudam a determinar o que queremos uma vez que são inviáveis de calcular. O método Naive Bayes propõe que devemos assumir que as palavras do texto não possuem correlação entre si (embora que isso não seja verdade no mundo real), levando a uma equação consideravelmente mais simples:

$$\begin{aligned}
\mathbb{P}(x_1, \ldots, x_n, S) &= \mathbb{P}(x_1 \mid x_2, x_3, \ldots, x_n, S)\mathbb{P}(x_2 \mid x_3, \ldots, x_n, S) \ldots \mathbb{P}(x_{n-1} \mid x_n, S)\mathbb{P}(x_n \mid S)\mathbb{P}(S) \\
&\approx \mathbb{P}(x_1 \mid S)\mathbb{P}(x_2 \mid S) \ldots \mathbb{P}(x_{n-1} \mid S)\mathbb{P}(x_n \mid S)\mathbb{P}(S) \\
&= \mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S)
\end{aligned}$$

O mesmo se aplica para o complemento de S:

$$\mathbb{P}(x_1, \ldots, x_n, H) \approx \mathbb{P}(H) \prod_{i=1}^{n} \mathbb{P}(x_i \mid H)$$

Naive Bayes: teoria

$$\mathbb{P}(x_1, \ldots, x_n, H) \approx \mathbb{P}(H) \prod_{i=1}^{n} \mathbb{P}(x_i \mid H)$$

Operar com a fórmula acima torna o processo muito mais conveniente pois cada P(xi|H) consiste de apenas:

$$\frac{\text{quantidade de xi na categoria}}{\text{quantidade de palavras na categoria}}$$

Enquanto P(H) consiste de:

$$\frac{\text{quantidade de emails na categoria H}}{\text{quantidade total de emails}}$$

Naive Bayes: teoria

Agora nossa nova equação tem o seguinte formato:

$$\mathbb{P}(S \mid x_1, \ldots, x_n) \approx \frac{\mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S)}{\mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S) + \mathbb{P}(H) \prod_{i=1}^{n} \mathbb{P}(x_i \mid H)}$$

Para computar a equação acima teremos que calcular as probabilidades de cada palavra, porém algumas palavras podem estar presentes em somente um dos grupos, spam ou não spam, levando a um dos elementos do multiplicando sendo substituído por zero e zerando todo o termo.

Uma consequência disso seria a possibilidade de um email spam que contém uma palavra que não estava em nenhum outro email spam ser automaticamente considerado não spam pois sua probabilidade seria zerada. Ou ainda um email com uma palavra que não está no conjunto de dados quebraria toda a equação com uma divisão por zero.

Naive Bayes: teoria

Portanto para todas as contas de probabilidade estaremos adicionando +1 no numerador, que é a quantidade de instâncias de uma dada palavra na categoria, e +2 no denominador, que é a quantidade de palavras na categoria.

A probabilidade de uma palavra w estar em uma dada categoria será:

$$\frac{(\text{quantidade de w na categoria} + 1)}{(\text{quantidade de palavras na categoria} + 2)}$$

Agora deveríamos poder simplesmente aplicar a fórmula e verificar que se a probabilidade se spam for maior que 0,5 o email é spam e caso contrário não é, porém estamos lidando com divisões extremamente pequenas que quando computadas podem facilmente levar a um underflow (quando o número é tão pequeno que o computador o transforma em zero) então teremos que fazer mais uma mudança na fórmula.

Naive Bayes: teoria

Porém antes de alterar a equação mais uma vez, é importante perceber que o que buscamos descobrir é o seguinte:

$$\frac{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(x_i \mid S)}{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(x_i \mid S)+\mathbb{P}(H)\prod_{i=1}^{n}\mathbb{P}(x_i \mid H)} > \frac{\mathbb{P}(H)\prod_{i=1}^{n}\mathbb{P}(x_i \mid H)}{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(x_i \mid S)+\mathbb{P}(H)\prod_{i=1}^{n}\mathbb{P}(x_i \mid H)}$$

E que para isso podemos desconsiderar o denominador e operar apenas sobre os numeradores para o caso de spam e não spam.

Portanto, considerando que a seguinte equação pode ser reduzida a zero, aplicaremos o seu logaritmo:

$$\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(x_i \mid S) \longrightarrow \log\left(\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(x_i \mid S)\right)$$

Naive Bayes: teoria

O logaritmo por sua vez pode ser reescrito como:

$$\log(\mathbb{P}(S)) + \sum_{i=1}^{n} \log(\mathbb{P}(x_i \mid S))$$

A partir disso já podemos inferir que se a desigualdade abaixo for verdadeira:

$$\log(\mathbb{P}(S)) + \sum_{i=1}^{n} \log(\mathbb{P}(x_i \mid S)) > \log(\mathbb{P}(H)) + \sum_{i=1}^{n} \log(\mathbb{P}(x_i \mid H))$$

A seguinte desigualdade também é verdade graças a propriedades de logaritmo:

$$\mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S) > \mathbb{P}(H) \prod_{i=1}^{n} \mathbb{P}(x_i \mid H)$$

E por fim concluímos que o email é spam.

Naive Bayes: teoria para notícias

Uma diferença entre a teoria apresentada até agora e a aplicação que faremos em seguida é que o conjunto de notícias possui 12 categorias.

Começaremos aplicando a seguinte fórmula em todas as categorias:

$$\log(\mathbb{P}(S)) + \sum_{i=1}^{n} \log(\mathbb{P}(x_i \mid S))$$

A lógica anterior ainda é válida porém não conseguiremos fixar um valor como 0,5 para associar a desigualdade, então escolheremos o maior valor dentre os calculados e a categoria associada a ele será nossa previsão.

Naive Bayes: prática I

A função abaixo computa a fórmula anterior para uma única notícia em uma única categoria, explicaremos as linhas no slide seguinte:

```
probabilidade <- function(noticia,dic,cords,total_notcs,notcs_tema){
  temp <- 0
  for(i in 1:length(noticia)){
    qtd <- cords[which(dic==noticia[i])]
    temp <- temp + log((qtd+1)/(notcs_tema+2))
  }
  return(log(notcs_tema/total_notcs)+temp)
}
```

Naive Bayes: prática I

`noticia` é um vetor de palavras (strings)

`dic` é o dicionário

`cords` é uma coluna da matriz de frequência de palavras por um tema específico

`total_notcs` quantidade de noticias

`notcs_tema` quantidade de noticias nesse tema

Temp receberá a soma dos logaritmos

```
temp <- 0
```

Para cada palavra extraímos sua quantidade

```
for(i in 1:length(noticia)){
qtd <- cords[which(dic==noticia[i])]
```

Naive Bayes: prática I

Temp recebe a soma dos logaritmos:

```
temp <- temp + log((qtd+1)/(notcs_tema+2))
```

Retornamos o restante da equação:

```
return(log(notcs_tema/total_notcs)+temp)
```

Lembrando que a equação é essa: $\log(\mathbb{P}(S)) + \sum_{i=1}^{n} \log(\mathbb{P}(x_i \mid S))$

Naive Bayes: prática II

A função a seguir invoca a função anterior para uma única notícia e todas as categorias:

```
scores <- function(noticia,cords,dicionario,geral){
  temas <- colnames(cords)
  probs <- c()
  for(i in 1:length(colnames(cords))){
    resul <- probabilidade(noticia,dicionario,cords[,i],
                           sum(geral[,2]==temas[i]),nrow(geral))
    probs <- c(probs,resul)
  }
  return(probs)
}
```

FIM.