

Aplicação de Técnicas de Aprendizagem de Máquina utilizando R

Prof. Mário de Noronha Neto

O material utilizado neste curso foi elaborado pelos professores Mario de Noronha Neto (IFSC) e Richard Demo Souza (UFSC)

Aprendizagem estatística - Classificação utilizando Naive Bayes



Classificadores baseados em métodos Bayesianos utilizam os dados de treinamento para calcular a probabilidade condicionada de cada saída em evidências fornecidas pelos valores das características. Desta forma, o classificador utiliza esta informação para predizer a classe mais provável para novos dados de entrada.

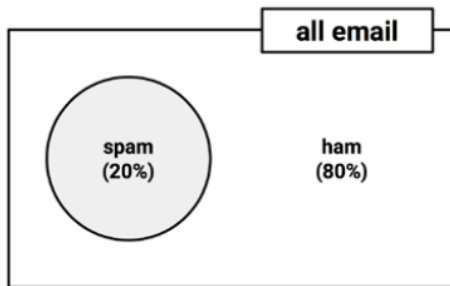
Tipicamente, esses classificadores são melhor aplicados em casos onde as informações de várias características devem ser consideradas simultaneamente para estimar a probabilidade total de um resultado. Esses métodos utilizam todas as evidências disponíveis para alterar sutilmente as previsões. Se um grande número de variáveis tiver efeitos relativamente pequenos, juntos, seu impacto combinado poderá afetar de forma significativa o resultado. Alguns exemplos de utilização:

- Classificação de textos, como por exemplo filtros de spam
- Detecção de anomalias ou de intrusos em redes de computadores

Conceitos básicos para métodos Bayesianos

A probabilidade de um evento ocorrer pode ser estimada dividindo o número de ocorrência de um evento pelo número de vezes que o experimento foi realizado.

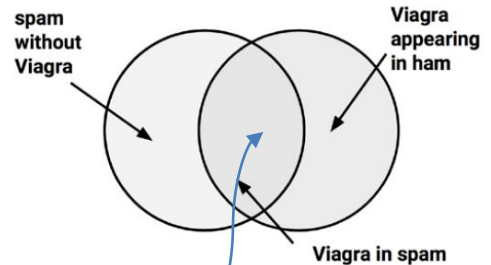
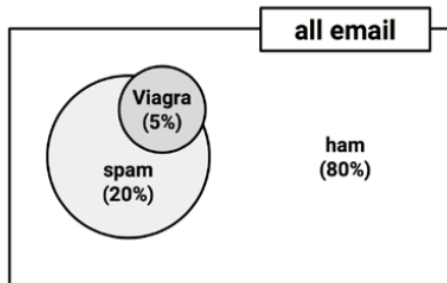
Probabilidade



Neste exemplo, os eventos 'spam' e 'não spam' são mutuamente excludentes, ou seja se $P(\text{spam}) = 0.2$, $P(\text{ham}) = 0.8$

Conceitos básicos para métodos Bayesianos

Probabilidade conjunta

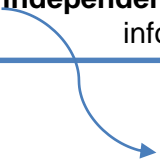


$$P(\text{spam} \cap \text{Viagra})$$

Probabilidade conjunta: Probabilidade de dois ou mais eventos ocorrerem simultaneamente

Conceitos básicos para métodos Bayesianos

Se dois eventos são totalmente descorrelacionados, eles são chamados de **eventos independentes**. Nesta situação, a ocorrência de um evento não traz informação alguma sobre o outro evento.


$$P(A \cap B) = P(A) * P(B)$$

Se todos os eventos são independentes, se torna impossível prever um evento com base nas observações de outros eventos. Em outras palavras, **eventos dependentes** são a base de modelos preditivos.

Conceitos básicos para métodos Bayesianos

Probabilidade condicional: Teorema de Bayes

Probabilidade condicional:
Probabilidade do evento **A**
ocorrer dado que o evento **B**
já ocorreu

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Probabilidade conjunta

Probabilidade de ocorrência
de B

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

Conceitos básicos para métodos Bayesianos

$$P(\text{spam}|\text{Viagra}) = \frac{P(\text{Viagra}|\text{spam})P(\text{spam})}{P(\text{Viagra})}$$

Diagram illustrating the components of the Bayesian formula:

- $P(\text{spam}|\text{Viagra})$: posterior probability
- $P(\text{Viagra}|\text{spam})$: likelihood
- $P(\text{spam})$: prior probability
- $P(\text{Viagra})$: marginal likelihood

	Viagra		
Frequency	Yes	No	Total
spam	4	16	20
ham	1	79	80
Total	5	95	100

	Viagra		
Likelihood	Yes	No	Total
spam	4 / 20	16 / 20	20
ham	1 / 80	79 / 80	80
Total	5 / 100	95 / 100	100

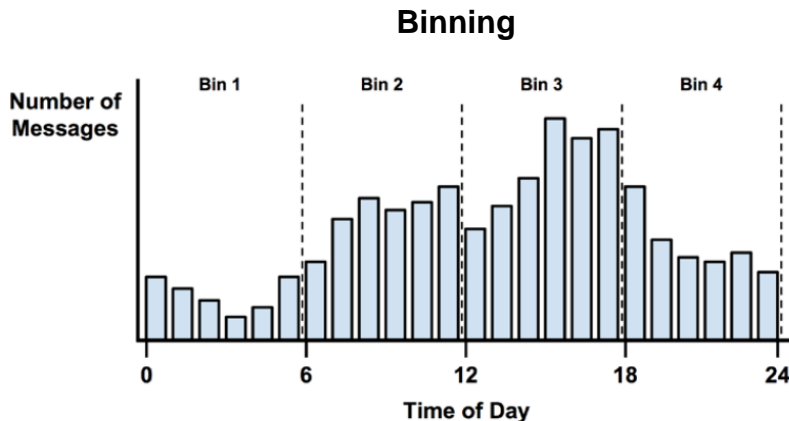
$$P(\text{Viagra} | \text{spam}) * P(\text{spam}) / P(\text{Viagra}) \text{ or } (4/20) * (20/100) / (5/100) = 0.80.$$

Este algoritmo assume que todas as características do *dataset* são igualmente importantes e **independentes**.

Utilizando características numéricas com o Naive Bayes



Como o Naive Bayes utiliza frequentemente tabelas para aprender os dados, cada característica precisa estar no formato categórico para que sejam criadas as combinações de classes e valores que compõe a matriz. Desta forma, os valores numéricos precisam ser representados por categorias.



Exemplo: Filtro de spam em SMS para celular



Passo 1: Coleta de dados

Dataset utilizado: <http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>

Este *dataset* inclui textos de mensagens SMS com um rótulo indicando se a mensagem é indesejada. Exemplos de spam e *ham* são mostrados no quadro abaixo.

Sample SMS ham	Sample SMS spam
<ul style="list-style-type: none">• Better. Made up for Friday and stuffed myself like a pig yesterday. Now I feel bleh. But, at least, its not writhing pain kind of bleh.• If he started searching, he will get job in few days. He has great potential and talent.• I got another job! The one at the hospital, doing data analysis or something, starts on Monday! Not sure when my thesis will finish.	<ul style="list-style-type: none">• Congratulations ur awarded 500 of CD vouchers or 125 gift guaranteed & Free entry 2 100 wkly draw txt MUSIC to 87066.• December only! Had your mobile 11mths+? You are entitled to update to the latest colour camera mobile for Free! Call The Mobile Update Co FREE on 08002986906.• Valentines Day Special! Win over £1000 in our quiz and take your partner on the trip of a lifetime! Send GO to 83600 now. 150 p/msg rcvd.

Passo 2: Explorando e preparando os dados

É necessário transformar as palavras e sentenças em uma forma que o computador possa entender. Os dados serão transformados em uma representação conhecida como **bag-of-words**

```
# read the sms data into the sms data frame
sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)

# examine the structure of the sms data
str(sms_raw)

'data.frame':   5559 obs. of  2 variables:
 $ type: chr   "ham" "ham" "ham" "spam" ...
 $ text: chr   "Hope you are having a good week. Just checking in"
         "K..give back my thanks." "Am also doing in cbe only. But have to
         pay." "complimentary 4 STAR Ibiza Holiday or £10,000 cash needs
         your URGENT collection. 09066364349 NOW from Landline not to lose
         out" | __truncated__ ...
```

Como é uma variável categórica,
é melhor converter para fator

Passo 2: Explorando e preparando os dados

```
# convert spam/ham to factor.  
sms_raw$type <- factor(sms_raw$type)  
  
# examine the type variable more carefully  
str(sms_raw$type)  
table(sms_raw$type)  
  
> str(sms_raw$type)  
Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...  
> table(sms_raw$type)  
ham spam  
4812 747
```

Passo 2: Explorando e preparando os dados

Os textos de SMS contêm palavras compostas, espaços, números e pontuações. Fazer este tipo de preparação manualmente é complexo. Este tipo de manipulação pode ser feito diretamente por um pacote disponível no R chamado *text mining* 'tm'

Cria uma coletânea de textos armazenada na memória a partir do vetor `sms_raw$text`

```
# build a corpus using the text mining (tm) package
install.packages("tm")
library(tm)
sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```

Passo 2: Explorando e preparando os dados

```
# examine the sms corpus  
print(sms_corpus)  
inspect(sms_corpus[1:2])
```

```
> print(sms_corpus)  
<<VCorpus>>  
Metadata: corpus specific: 0, document level (indexed): 0  
Content: documents: 5559
```

```
> inspect(sms_corpus[1:2])  
<<VCorpus>>  
Metadata: corpus specific: 0, document level (indexed): 0  
Content: documents: 2
```

```
[[1]]  
<<PlainTextDocument>>  
Metadata: 7  
Content: chars: 49
```

```
[[2]]  
<<PlainTextDocument>>  
Metadata: 7  
Content: chars: 23
```

Passo 2: Explorando e preparando os dados

Visualizando as mensagens

```
as.character(sms_corpus[[1]])  
lapply(sms_corpus[1:2], as.character)  
  
> as.character(sms_corpus[[1]])  
[1] "Hope you are having a good week. Just checking in"  
  
> lapply(sms_corpus[1:2], as.character)  
$`1`  
[1] "Hope you are having a good week. Just checking in"  
  
$`2`  
[1] "K..give back my thanks."
```

Passo 2: Explorando e preparando os dados

Limpeza dos dados

```
# clean up the corpus using tm_map()
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))

# show the difference between sms_corpus and corpus_clean
as.character(sms_corpus[[1]])
as.character(sms_corpus_clean[[1]])

> sms_corpus_clean <- tm_map(sms_corpus,
  content_transformer(tolower))

> as.character(sms_corpus[[1]])
[1] "Hope you are having a good week. Just checking in"
> as.character(sms_corpus_clean[[1]])
[1] "hope you are having a good week. just checking in"
```


Passo 2: Explorando e preparando os dados

Limpeza dos dados

```
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers) # remove numbers  
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords()) # remove stop words  
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation) # remove punctuation  
  
# illustration of word stemming  
install.packages("snowballc")  
library(snowballc)  
wordStem(c("learn", "learned", "learning", "learns"))  
[1] "learn"  "learn"  "learn"  "learn"  
  
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)  
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace) # eliminate unneeded whitespace
```

Passo 2: Explorando e preparando os dados

```
# examine the final clean corpus
lapply(sms_corpus[1:3], as.character)
lapply(sms_corpus_clean[1:3], as.character)
```

SMS messages before cleaning	SMS messages after cleaning
<pre>> as.character(sms_corpus[1:3]) [[1]] Hope you are having a good week. Just checking in [[2]] K..give back my thanks. [[3]] Am also doing in cbe only. But have to pay.</pre>	<pre>> as.character(sms_corpus_clean[1:3]) [[1]] hope good week just check [[2]] kgive back thank [[3]] also cbe pay</pre>

Passo 2: Explorando e preparando os dados

Transformando textos em palavras: A função *DocumentTermMatrix* () transforma uma coletânea em uma estrutura de dados chamada *Document Term matrix (DTM)* em que as linhas indicam as mensagens (SMS) e as colunas os termos (palavras)

```
# create a document-term sparse matrix
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
<<DocumentTermMatrix (documents: 5559, terms: 6518)>>
Non-/sparse entries: 42113/36191449
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
```

message #	balloon	balls	bam	bambling	band
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Passo 2: Explorando e preparando os dados

Criando conjuntos de dados para treinamento e teste

```
# creating training and test datasets
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]

# also save the labels
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type

# check that the proportion of spam is similar
prop.table(table(sms_train_labels))
prop.table(table(sms_test_labels))
> prop.table(table(sms_train_labels))
      ham      spam
0.8647158 0.1352842
> prop.table(table(sms_test_labels))
      ham      spam
0.8683453 0.1316547
```

Visualizando os dados: Nuvem de palavras

can get call now just free lor say claim said late gonna also buy pick guy box wan person custom

Passo 2: Explorando e preparando os dados

Visualizando os dados: Nuvem de palavras

```
# subset the training data into spam and ham groups
spam <- subset(sms_raw, type == "spam")
ham <- subset(sms_raw, type == "ham")

wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



Passo 2: Explorando e preparando os dados

Removendo palavras que aparece com menos frequência

```
# indicator features for frequent words
findFreqTerms(sms_dtm_train, 5)

# save frequently-appearing terms to a character vector
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)

chr [1:1136] "abiola" "abl" "abt" "accept" "access" "account"
"across" "act" "activ" ...

# create DTMs with only the frequent terms
sms_dtm_freq_train <- sms_dtm_train[, sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[, sms_freq_words]
```

Palavras que aparecem menos de 5 vezes na sequência de treinamento (aprox. 0.1%)

```
 sms_dtm_freq_train      Large DocumentTermMatrix (6 elements, 682.3 Kb)
  i : int [1:24950] 1 1 1 1 1 2 2 3 3 4 ...
  j : int [1:24950] 152 391 441 490 1084 70 972 31 708 139 ...
  v : num [1:24950] 1 1 1 1 1 1 1 1 1 1 ...
 nrow : int 4169
 ncol : int 1139
```

Passo 2: Explorando e preparando os dados

O Naive Bayes tipicamente é treinado com variáveis categóricas. As células da matriz esparsa são numéricas e medem o número de vezes que uma palavra aparece em uma mensagem. Desta forma, é necessário converter essas variáveis para categóricas, indicando *sim* ou *não*, dependendo se a palavra aparece ou não na mensagem.

```
# convert counts to a factor
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}

# apply() convert_counts() to columns of train/test data
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```


Passo 3: Treinando o modelo

Para a classificação utilizamos o Naive Bayes, utilizaremos a função *naiveBayes* () do pacote 'e1071'

Naive Bayes classification syntax

using the *naiveBayes* () function in the *e1071* package

Building the classifier:

```
m <- naiveBayes(train, class, laplace = 0)
```

- **train** is a data frame or matrix containing training data
- **class** is a factor vector with the class for each row in the training data
- **laplace** is a number to control the Laplace estimator (by default, 0)

The function will return a naive Bayes model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

- **m** is a model trained by the *naiveBayes* () function
- **test** is a data frame or matrix containing test data with the same features as the training data used to build the classifier
- **type** is either "class" or "raw" and specifies whether the predictions should be the most likely class value or the raw predicted probabilities

The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the **type** parameter.

Example:

```
sms_classifier <- naiveBayes(sms_train, sms_type)  
sms_predictions <- predict(sms_classifier, sms_test)
```

Passo 3: Treinando o modelo

```
## Step 3: Training a model on the data ----
install.packages("e1071")
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```

sms_classifier      Large naiveBayes (4 elements, 1.4 Mb)
apriori: 'table' int [1:2(1d)] 3605 564
.. attr(*, "dimnames")=List of 1
.. ..$ sms_train_labels: chr [1:2] "ham" "spam"
tables :List of 1139
..$ abiola : table [1:2, 1:2] 0.99806 1 0.00194 0
.. .. attr(*, "dimnames")=List of 2
.. .. ..$ sms_train_labels: chr [1:2] "ham" "spam"
.. .. ..$ abiola : chr [1:2] "No" "Yes"
..$ abl : table [1:2, 1:2] 0.99473 1 0.00527 0
.. .. attr(*, "dimnames")=List of 2
.. .. ..$ sms_train_labels: chr [1:2] "ham" "spam"
.. .. ..$ abl : chr [1:2] "No" "Yes"
..$ abt : table [1:2, 1:2] 0.99584 1 0.00416 0
.. .. attr(*, "dimnames")=List of 2
.. .. ..$ sms_train_labels: chr [1:2] "ham" "spam"
.. .. ..$ abt : chr [1:2] "No" "Yes"
..$ accept : table [1:2, 1:2] 0.99861 1 0.00139 0
.. .. attr(*, "dimnames")=List of 2
.. .. ..$ sms_train_labels: chr [1:2] "ham" "spam"
.. .. ..$ accept : chr [1:2] "No" "Yes"
..$ access : table [1:2, 1:2] 0.999723 0.992908 0.000277 0.007092
.. .. attr(*, "dimnames")=List of 2
.. .. ..$ sms_train_labels: chr [1:2] "ham" "spam"
.. .. ..$ access : chr [1:2] "No" "Yes"
```

Passo 4: Avaliando o desempenho do modelo



```
## Step 4: Evaluating model performance ----  
sms_test_pred <- predict(sms_classifier, sms_test)  
  
library(gmodels)  
CrossTable(sms_test_pred, sms_test_labels,  
            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,  
            dnn = c('predicted', 'actual'))
```

Total Observations in Table: 1390

predicted	actual		Row Total
	ham	spam	
ham	1201 0.995	30 0.164	1231
spam	6 0.005	153 0.836	159
Column Total	1207 0.868	183 0.132	1390

Realizar testes para outras configurações

- Variar a proporção das sequências de treinamento e teste
- Variar o estimador Laplace
- Alterar parâmetros no pré-processamento, como por exemplo variar o valor da função '*findFreqTerms*'
- Testar o método com outro conjunto de dados. Utilizar o arquivo '*Social_Network_Ads.csv*' extraído do site:
<https://www.superdatascience.com/machine-learning/>