Apêndice II: Análise de Sentimentos com ML no R - Random Forest

Avaliações de uma Loja Virtual

Banco de Dados

Será usado um banco de dados do **Kaggle** que possui avaliações escritas pelos consumidores de um *e-commerce* de roupas femininas. O objetivo é prever se o cliente vai ou não recomendar o produto com base no texto da avaliação. Ou seja, a recomendação do cliente seria o equivalente a um sentimento positivo com relação ao produto e à loja.

```
library(readr)
library(dplyr)
library(tm)
```

O tratamento dos textos será feito por meio do pacote `tm` que possui uma gama de ferramentas disponíveis para *Text Mining*.

```
t1 <- read_csv("Womens Clothing E-Commerce Reviews.csv")
t1 <- t1[1:800,]
glimpse(t1)
```

```
## Rows: 800
## Columns: 11
## $ X1                         <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,...
## $ `Clothing ID`              <dbl> 767, 1080, 1077, 1049, 847, 1080, 858, 85...
## $ Age                        <dbl> 33, 34, 60, 50, 47, 49, 39, 39, 24, 34, 5...
## $ Title                      <chr> NA, NA, "Some major design flaws", "My fa...
## $ `Review Text`              <chr> "Absolutely wonderful - silky and sexy an...
## $ Rating                     <dbl> 4, 5, 3, 5, 5, 2, 5, 4, 5, 5, 3, 5, 5, 5,...
## $ `Recommended IND`          <dbl> 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,...
## $ `Positive Feedback Count` <dbl> 0, 4, 0, 0, 6, 4, 1, 4, 0, 0, 14, 2, 2, 0...
## $ `Division Name`            <chr> "Initmates", "General", "General", "Gener...
## $ `Department Name`          <chr> "Intimate", "Dresses", "Dresses", "Bottom...
## $ `Class Name`               <chr> "Intimates", "Dresses", "Dresses", "Pants...
```

O banco possui diversas informações como a idade do cliente, ID do produto, título da avaliação, texto da avaliação, etc. Entretanto, com o objetivo de exemplificar o processos de análise de sentimentos (textual, no caso), serão filtrados apenas o texto da avaliação e o campo de classificação relacionada ao sentimento, ou seja, a variável indicadora de recomendação.

Por fim de otimização do código e apenas para demonstração, será utilizado apenas uma parte do banco de dados.

```r
t1 <- t1 %>% select(`Review Text`, `Recommended IND`)
```

Pré-Processamento para Modelagem

Passo 1: Criação do Corpus

A variável que contém os textos precisa ser convertida em um Corpus, que é uma coleção de documentos/textos.

```r
corpus = Corpus(VectorSource(t1$`Review Text`))
corpus[[1]][1]
```

```
## $content
## [1] "Absolutely wonderful - silky and sexy and comfortable"
```

```r
t1$`Recommended IND`[1]
```

```
## [1] 1
```

Acima é possível ver um texto de avaliação, já dentro do Corpus, e sua classificação sentimental. Claramente há uma grande satisfação do cliente expressa no texto e, por isso, o produto foi recomendado.

Passo 2: Conversão para *lowercase*

O modelo precisa tratar palavras como 'Comfortable' e 'comfortable' da mesma maneira. Por isso, todas as palavras serão convertidas para *lowercase*.

```r
corpus = tm_map(corpus, PlainTextDocument)
corpus = tm_map(corpus, tolower)
corpus[[1]][1]
```

```
## $content
## [1] "absolutely wonderful - silky and sexy and comfortable"
```

Passo 3: Remoção de Pontuações

A ideia neste passo é remover tudo que não é letra ou número.

```r
corpus = tm_map(corpus, removePunctuation)
corpus[[1]][1]
```

```
## $content
## [1] "absolutely wonderful  silky and sexy and comfortable"
```

Passo 4: Remoção de *Stopwords*

A remoção de *stopwords* é muito importante pois são termos que não possuem contribuição sentimental e são muito frequentes nos Corpus.

```
corpus = tm_map(corpus, removeWords, c("cloth", stopwords("english")))
corpus[[1]][1]
```

```
## $content
## [1] "absolutely wonderful  silky  sexy  comfortable"
```

Foram removidas as *stopword* do inglês, além da palavra 'cloth', uma vez que o banco diz respeito a avaliações de roupas e esta palavra em específico não aumentará o poder preditivo do modelo por não possuir sentido sentimental e ser muito frequente.

Passo 5: Stemming

O objetivo do Stemming é reduzir o número de formas flexionais de palavras que aparecem no texto, selecionando apenas o núcleo das palavras.

```
corpus = tm_map(corpus, stemDocument)
corpus[[1]][1]
```

```
## $content
## [1] "absolut wonder silki sexi comfort"
```

Criação da Matriz de Termos do Documento

Após completadas as principais etapas do pré-processamento para modelagem, serão extraídas as frequências de cada palavra que serão utilizadas como *features* do modelo preditivo.

```
frequencies = DocumentTermMatrix(corpus)
```

Cada linha corresponde a uma avaliação e as colunas se referem às frequências das palavras em cada avaliação.

Isso resulta em uma matriz que contém muitos zeros, um problema chamado esparsidade (*sparsity*). É aconselhado remover palavras que contenham muitos zeros, ou seja, pouco frequentes nos textos.

```
sparse = removeSparseTerms(frequencies, 0.995)
dim(frequencies)
```

```
## [1]  800 2516
```

```
dim(sparse)
```

```
## [1] 800 695
```

Houve uma grande redução das colunas da matriz, de forma a selecionar apenas as palavras mais frequentes. Foi passado para a função que permaneçam apenas termos com esparsidade máxima de 0.995.

O passo final de preparação dos dados para a modelagem é converter a matriz em um *dataframe*, que é o formato mais utilizado no *R* para modelagem preditiva.

```r
tSparse = as.data.frame(as.matrix(sparse))
colnames(tSparse) = make.names(colnames(tSparse))
tSparse$recommended_id = t1$`Recommended IND`
```

```r
tSparse[1,tSparse[1,]>0]
```

```
##   absolut comfort sexi wonder recommended_id
## 1       1       1    1      1              1
```

Modelagem

Antes da modelagem em si, é importante verificar a distribuição das classes da variável resposta no banco de dados.

```r
prop.table(table(tSparse$recommended_id))
```

```
##
##       0       1
## 0.16875 0.83125
```

É possível ver que 83.125% das avaliações são de consumidores que recomendaram o produto.

Criação das Bases de Treino e Teste do Modelo

Para poder avaliar o poder preditivo do modelo, os dados serão divididos em treino e teste. A partição será feita por meio do pacote `caTools`. Os dados serão repartidos em 70% para treino do modelo e 30% para teste do mesmo.

```r
library(caTools)
set.seed(100)
split = sample.split(tSparse$recommended_id, SplitRatio = 0.7)
trainSparse = subset(tSparse, split==TRUE)
testSparse = subset(tSparse, split==FALSE)
```

Random Forest

Será utilizado o algoritmo *Random Forest* que é uma coleção de várias árvores de classificação que operam em conjunto. Para isso, será usado o pacote `randomForest`. Antes de ajustar o modelo é importante assegurar que a variável resposta é da classe fator.

```r
library(randomForest)
set.seed(100)
trainSparse$recommended_id = as.factor(trainSparse$recommended_id)
testSparse$recommended_id = as.factor(testSparse$recommended_id )

# Ajuste do Modelo
RF_model = randomForest(recommended_id ~ ., data=trainSparse)
```

Como foi utilizado o método de *Random Forest* é possível calcular a importância dos preditores e detectar, então, os termos mais importantes na distinção das classes.

```
((importance(RF_model)) %>% as.data.frame() %>% arrange(desc(MeanDecreaseGini))) %>%
  head()
```

```
##            MeanDecreaseGini
## return             3.710075
## disappoint         3.024281
## back               2.715741
## made               1.748764
## zipper             1.605278
## look               1.461896
```

Avaliação do Modelo

```
predictRF = predict(RF_model, newdata=testSparse)

conf.matrix = table(Actual=testSparse$recommended_id, Predict=predictRF)
conf.matrix
```

```
##       Predict
## Actual   0   1
##      0   3  38
##      1   0 200
```

```
n = sum(conf.matrix) # number of instances
nc = nrow(conf.matrix) # number of classes
diag = diag(conf.matrix) # number of correctly classified instances per class
rowsums = apply(conf.matrix, 1, sum) # number of instances per class
colsums = apply(conf.matrix, 2, sum) # number of predictions per class
p = rowsums / n # distribution of instances over the actual classes
q = colsums / n # distribution of instances over the predicted classes

# Accuracy
sum(diag) / n
```

```
## [1] 0.8423237
```

```
precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)
data.frame(precision, recall, f1)
```

```
##   precision     recall        f1
## 0 1.0000000 0.07317073 0.1363636
## 1 0.8403361 1.00000000 0.9132420
```

É possível verificar que o modelo tem alto poder preditivo para identificar avaliações que resultarão em recomendação. Contudo, para as demais, o poder preditivo do modelo cai consideravelmente. Este problema foi impulsionado pelo desbalanceamento das classes. Técnicas para corrigir problemas de desbalanceamento poderiam melhorar o resultado obtido.

De forma análoga é possível aplicar diversos outros algoritmos de *machine learning* como Naive Bayes, SVM, etc.