Classificação de Tumor Benigno ou Maligno

Classificação com k-NN no R

César Lemos - B.S. in Economics, Mathematics and Data Scientist

11/02/2020

Abstract

k-NN, ou k Nearest Neighbors, é um modelo de classificação simples, porém eficiente em sua grande maioria das vezes, que visa clusterizar variáveis de acordo com suas características calculando a distância (normalmente Euclidiana) entre elas e agrupando-as por proximidade. No exemplo a seguir será construído um modelo de predição de Câncer de Próstata com a finalidade de verificar se é Benigno ou Maligno.

Contents

1 Introdução

O classificador k-NN é definido pelas características de exemplos não-rotulados e atribui uma classe definida de exemplos rotulados semelhantes. Estas características tem suas proximidades calculadas através de métricas como a Distância Euclidiana, Distância de Hamming, similaridade do Cosseno, dentre outras. Apesar de simples, o classificador k-NN é usado com sucesso para recomendação de músicas, identificação de padrões nos dados genéticos, dentre outras.

Em resumo, o k-NN é um algorítmo simples, efetivo e possui uma rápida fase de treino. Entretanto, pode-se destacar os seguintes pontos negativos:

- Não produz um modelo, limitando a capacidade de entender como os recursos estão relacionados à classe (como um modelo de regressão faz, por exemplo);
- Requer a seleção de um K inicial apropriado;
- Recursos nominais e dados ausentes requer processamento adicional.

1.1 Link para o dataset

https://www.kaggle.com/hdza1991/breast-cancer-wisconsin-data-set

2 Biblioteca utilizada

```
library(tidyverse)
library(caret)
library(class)
```

3 Importando os dados

```
dt <- read_csv(choose.files())[,-1]
dt$diagnosis <- factor(dt$diagnosis, levels = c("B","M"))
```

4 Analise Exploratória dos dados

4.1 Verificando os tipos dos dados e ausência de valores

```
# Verificando as primeiras linhas da tabela
head(dt)
```

```
## # A tibble: 6 x 31
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
##   <fct>           <dbl>        <dbl>          <dbl>     <dbl>           <dbl>
## 1 M                18.0         10.4          123.      1001           0.118
## 2 M                20.6         17.8          133.      1326           0.0847
## 3 M                19.7         21.2          130       1203           0.110
```

```
## 4 M               11.4         20.4           77.6      386.          0.142
## 5 M               20.3         14.3          135.      1297           0.100
## 6 M               12.4         15.7           82.6      477.          0.128
## # ... with 25 more variables: compactness_mean <dbl>, concavity_mean <dbl>,
## #   `concave points_mean` <dbl>, symmetry_mean <dbl>,
## #   fractal_dimension_mean <dbl>, radius_se <dbl>, texture_se <dbl>,
## #   perimeter_se <dbl>, area_se <dbl>, smoothness_se <dbl>,
## #   compactness_se <dbl>, concavity_se <dbl>, `concave points_se` <dbl>,
## #   symmetry_se <dbl>, fractal_dimension_se <dbl>, radius_worst <dbl>,
## #   texture_worst <dbl>, perimeter_worst <dbl>, area_worst <dbl>,
## #   smoothness_worst <dbl>, compactness_worst <dbl>, concavity_worst <dbl>,
## #   `concave points_worst` <dbl>, symmetry_worst <dbl>,
## #   fractal_dimension_worst <dbl>
```

Dados ausentes podem ser um problema para modelos como k-NN. Caso exista, é necessário tratar de acordo. Se o dataset for pequeno, é recomendável realizar um processo de interpolação para preencher os valores ausentes. Caso o dataset seja grande e exista poucos casos, pode excluir as linhas com miss value. No caso deste dataset, não existe valores ausentes conforme mostra a função abaixo.

```r
# Verificando a existência de miss value
sapply(dt, function (x) sum(is.na(x)))
```

```
##               diagnosis             radius_mean            texture_mean
##                       0                       0                       0
##         perimeter_mean               area_mean         smoothness_mean
##                       0                       0                       0
##        compactness_mean          concavity_mean     concave points_mean
##                       0                       0                       0
##           symmetry_mean  fractal_dimension_mean               radius_se
##                       0                       0                       0
##              texture_se            perimeter_se                 area_se
##                       0                       0                       0
##           smoothness_se          compactness_se            concavity_se
##                       0                       0                       0
##       concave points_se             symmetry_se    fractal_dimension_se
##                       0                       0                       0
##            radius_worst           texture_worst         perimeter_worst
##                       0                       0                       0
##              area_worst        smoothness_worst       compactness_worst
##                       0                       0                       0
##         concavity_worst    concave points_worst          symmetry_worst
##                       0                       0                       0
## fractal_dimension_worst
##                       0
```

4.2 Verificando quantidade da variável dependente no dataset

```
# Total de Benignos e Malígnos
table(dt$diagnosis)
```

```
##
##   B   M
## 357 212
```

```
# Benignos e Malígnos em %
round(prop.table(table(dt$diagnosis))*100, digits = 1)
```

```
##
##    B    M
## 62.7 37.3
```

4.3 Normalizando as variáveis quantitativas

Variáveis quantitativas podem ser um problema se estiverem em escalas muito diferentes entre si. Isso porque um valor muito alto, com uma alta amplitude, pode enviesar toda a classificação e resultar em um modelo pouco eficiente. Para evitar que isto ocorra, geralmente é normalizado todas as variáveis quantitativas, trazendo-as para uma mesma unidade de medida.

Existem algumas formas de normalizar estas variáveis. Nesta publicação será abordada duas formas: a z-score e a normalização min-max.

4.3.1 Normalização Min-Max

Esta normalização é calculada da seguinte forma:

$x_{ajust} = (x - min(x))/(max(x) - min(x))$

```
norm.minmax <- function(x) {
    return((x - min(x)) / (max(x) - min(x)))
  }
```

4.3.2 Normalização Z-score

Esta normalização é calculada da seguinte forma:

$x_{ajust} = (x - mean(x))/sd(x)$

Onde $sd(x)$ é o **desvio padrão** dos dados observados.

```
norm.sd <- function(x){
    return((x - mean(x))/sd(x))
  }
```

4.3.3 Normalizando os dados

Para fins de comparação, o k-NN vai ser rodado com os dados normalizados pelos dois métodos explicados anteriormente

```
dt_norm1 <- as.data.frame(lapply(dt[,2:31], norm.minmax))
dt_norm2 <- as.data.frame(lapply(dt[,2:31], norm.sd))
```

5 Construíndo o modelo de Classificação k-NN

5.1 Criando dataset de treino e teste

Dataset de treino e teste utilizando a normalização min-max.

```
train1 <- dt_norm1[1:469,]
test1 <- dt_norm1[470:569,]
```

Dataset de treino e teste utilizando o Z-score para normalização dos dados.

```
train2 <- dt_norm2[1:469,]
test2 <- dt_norm2[470:569,]
```

Criando as labels de saída.

```
label_train <- dt[1:469,1]
label_test <- dt[470:569,1]
```

5.2 Criando o modelo k-NN

Uma sugestão acadêmica para a escolha do **k** é calcular a raiz quadrada do tamanho da amostra e usar o valor obtido. Neste caso, a raiz de 469 é, aproximadamente, 22.

```
# Modelo com Norm. min-max
dt_pred1 <- knn(train = train1, test = test1, cl = label_train$diagnosis, k = 22)

# Modelo normalizado com Z-score
dt_pred2 <- knn(train = train2, test = test2, cl = label_train$diagnosis, k =22)
```

5.2.1 Matriz de Confusão usando o modelo normalizado com min-max.

```
confusionMatrix(dt_pred1, label_test$diagnosis)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 77  2
##          M  0 21
##
##                Accuracy : 0.98
##                  95% CI : (0.9296, 0.9976)
##     No Information Rate : 0.77
##     P-Value [Acc > NIR] : 2.106e-09
##
##                   Kappa : 0.9418
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9130
##          Pos Pred Value : 0.9747
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7700
##          Detection Rate : 0.7700
##    Detection Prevalence : 0.7900
##       Balanced Accuracy : 0.9565
##
##        'Positive' Class : B
##
```

5.2.2 Matriz de Confusão usando o modelo normalizado com Z-score.

```
confusionMatrix(dt_pred2, label_test$diagnosis)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 77  2
##          M  0 21
##
##                Accuracy : 0.98
##                  95% CI : (0.9296, 0.9976)
##     No Information Rate : 0.77
##     P-Value [Acc > NIR] : 2.106e-09
##
##                   Kappa : 0.9418
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9130
##          Pos Pred Value : 0.9747
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7700
##          Detection Rate : 0.7700
##    Detection Prevalence : 0.7900
##       Balanced Accuracy : 0.9565
##
##        'Positive' Class : B
##
```

6 Conclusão

Ambos os modelos apresentaram uma performance excelente, não havendo nenhuma diferença entre eles. Os modelos obtiveram uma acurácia de 98%, com apenas 2 casos de falso positivo, o que gerou uma especificidade de 91,3%.

Uma sugestão para verificar se o **k** escolhido foi o melhor é rodar o modelo com outros valores para **k** e gravar os resultados para uma comparação de sensibilidade, especificidade e acurácia.