Lista 01 - Aprendizado de Máquinas

18 junho, 2023

Exercício 01

Faça uma pesquisa sobre aplicações práticas de aprendizado de máquinas e descreva aplicações de cada um dos problemas abaixo, incluindo a descrição dos vetores de características, rótulos ou respostas quando apropriado para cada um dos problemas:

a) Problema de classificação. R.: Este método é utilizado para categorizar objetos em rótulo pré-determinados. Um exemplo clássico de aplicação é a detecção de spam, no qual a entrada é um e-mail e os rótulos são "spam" ou "não spam". Nesse caso, o vetor de características pode ser composto por fatores como remetente, assunto do e-mail, tamanho da mensagem, frequência de envio e destinatários.

b) Problema de regressão. R.: Este método avalia relações e dependência entre variáveis. Um exemplo de aplicação seria predizer o valor de um imóvel. A partir de características (vetor de características) como tamanho, localização, número de cômodos, garagem e "idade" (data de construção), pode-se construir um modelo que determina o valor do imóvel (resposta).

c) Problema de Agrupamento. R.: Este é um método de aprendizado não-supervisionado, no qual conhecemos somente as características dos objetos. Dessa forma, os elementos são agrupados em clusters de acordo com sua similaridade. Um exemplo seria um modelo de classificação de imagens que, a partir de um conjunto de imagens de elefantes, os classifica entre asiático e africano (clusters). Esse modelo poderia fazer a clusterização a partir de características como tamanho do animal, formato das orelhas e aparência das presas.

Exercício 02

Descreva com suas próprias palavras o que é a "maldição da dimensionalidade".

A maldição da dimensionalidade é um problema observado em métodos que dependem de vizinhança. Pode-se dizer que, quanto maior a dimensão do modelo, mais longe fica o vizinho mais próximo (dados mais dispersos), dificultando consideravelmente o processo de identificação de padrões e afetando o desempenho do modelo.

Exercício 03

Implemente o método dos vizinhos mais próximos para um problema de classificação. Sua função deve receber como argumento:

1) Um número k representando o número de vizinhos para usar no kNN;

2) Um vetor $x = (x_1, x_2)$ com duas componentes;

3) Uma *dataframe* D com colunas x_1, x_2 e y, em que x_1 e x_2 são valores numéricos representando duas componentes do vetor de características x (i.e., $x = (x_1, x_2)$) e y é um fator representando os rótulos dos pontos. Abaixo tem-se um exemplo dessa *dataframe* com duas classes:

```r
D <- tibble( x_1 = rnorm(100,1,1),
             x_2 = rnorm(100,-1,2),
             y   = factor(sample(c("one","two","three"),100,replace = T)))
head(D)
```

x_1	x_2	y
0.7693256	-2.1821482	three
1.5858593	-7.1042952	one
1.1039001	0.2130045	two
0.6285294	-4.7242336	three
0.8307628	-1.3644112	three
0.3145038	0.8313677	three

A função deve ter a assinatura `function(k,x,D)` e deve retornar a classe mais provável associada ao ponto x.

dica: Você pode fazer o kNN usando uma sequencia de comandos encadeados pelo operador pipe `%>%`. Por exemplo, teste a seguinte sequencia de comandos com a *dataframe* D anterior:

```r
x = c(1,2)
k = 10
D2 <- D %>%
  mutate( dist = (x[1] - x_1)^2 + (x[2] - x_2)^2 ) %>%
  arrange( dist ) %>% head(k) %>% count(y)
```

```r
## RESOLUÇÃO DO EXERCÍCIO 03

knn_func <- function(k, x, D) {
    valor_predito <- D %>%
    # calculando as distancias Euclideanas para cada ponto em D
    mutate(dist = sqrt((x[1] - x_1)^2 + (x[2] - x_2)^2)) %>%
    # ordenando os resuldados obtidos em ordem crescente
    arrange(dist) %>%
    # separando os rótulos dos k pontos mais próximos
    slice(1:k) %>%
    # contando a quantidade de cada rótulo obtido para k vizinhos
    count(y) %>%
    # identificando qual dos rótulos foi mais frequente
    slice_max(n, with_ties = FALSE) %>%
    pull(y)
  return(valor_predito)
}

## TESTANDO A CLASSIFICACAO

k <- 10
x <- c(1.5, -0.5)
D <- tibble( x_1 = rnorm(150,5,3),
             x_2 = rnorm(150,-5,3),
             y   = factor(sample(c("vermelho","verde","azul"),150,replace = T)))
predicao <- knn_func(k, x, D)
print(paste("Valor predito:", predicao))
```

```
## [1] "Valor predito: verde"
```

Exercício 04

Usando o banco de dados `iris` e sua implementação do kNN do exercício anterior, calcule quantos pontos são classificados corretamente de acordo com o rótulo `Species` usando as colunas `Petal.length` e `Sepal.length` com $k = 10$ e com $k = 1$.

dica 1: Você pode carregar o banco Iris no R da seguinte forma:

```
library(tidyverse)
data("iris") # Carrega o banco no ambiente global
iris <- as_tibble(iris) %>%  # Converte para a dataframe tibble
  select(Petal.Length,Sepal.Length,Species) %>% # Seleciona colunas da dataframe
  rename( x_1 = Petal.Length, x_2 = Sepal.Length, y = Species) # Renomeia as colunas

head(iris)
```

x_1	x_2	y
1.4	5.1	setosa
1.4	4.9	setosa
1.3	4.7	setosa
1.5	4.6	setosa
1.4	5.0	setosa
1.7	5.4	setosa

dica 2: As funções `map` da biblioteca `purrr` do pacote *tidyverse* são muito úteis! Por exemplo, a função `pmap_lgl` aplica uma função à argumentos fornecidos por uma lista e retorna os resultados da função como um vetor de booleanos (neste caso, a função deve retornar valores booleanos). Rode o exemplo abaixo:

```
l_iris <- as.list(iris) # converte a dataframe em uma lista
                        # que possui elementos com nomes x_1, x_2 e y

# Aplica a função usando os valores dados pelos valores na lista,
#  concatena os resultados da função em um vetor booleano.
v_bool <- pmap_lgl(l_iris, function(x_1,x_2,y){
          print(str_c(x_1,", ",x_2,", ",y))
          return(y == "setosa")
})
```

```
## [1] "1.4, 5.1, setosa"
## [1] "1.4, 4.9, setosa"
## [1] "1.3, 4.7, setosa"
## [1] "1.5, 4.6, setosa"
## [1] "1.4, 5, setosa"
## [1] "1.7, 5.4, setosa"
## [1] "1.4, 4.6, setosa"
## [1] "1.5, 5, setosa"
## [1] "1.4, 4.4, setosa"
## [1] "1.5, 4.9, setosa"
## [1] "1.5, 5.4, setosa"
```

```
## [1] "1.6, 4.8, setosa"
## [1] "1.4, 4.8, setosa"
## [1] "1.1, 4.3, setosa"
## [1] "1.2, 5.8, setosa"
## [1] "1.5, 5.7, setosa"
## [1] "1.3, 5.4, setosa"
## [1] "1.4, 5.1, setosa"
## [1] "1.7, 5.7, setosa"
## [1] "1.5, 5.1, setosa"
## [1] "1.7, 5.4, setosa"
## [1] "1.5, 5.1, setosa"
## [1] "1, 4.6, setosa"
## [1] "1.7, 5.1, setosa"
## [1] "1.9, 4.8, setosa"
## [1] "1.6, 5, setosa"
## [1] "1.6, 5, setosa"
## [1] "1.5, 5.2, setosa"
## [1] "1.4, 5.2, setosa"
## [1] "1.6, 4.7, setosa"
## [1] "1.6, 4.8, setosa"
## [1] "1.5, 5.4, setosa"
## [1] "1.5, 5.2, setosa"
## [1] "1.4, 5.5, setosa"
## [1] "1.5, 4.9, setosa"
## [1] "1.2, 5, setosa"
## [1] "1.3, 5.5, setosa"
## [1] "1.4, 4.9, setosa"
## [1] "1.3, 4.4, setosa"
## [1] "1.5, 5.1, setosa"
## [1] "1.3, 5, setosa"
## [1] "1.3, 4.5, setosa"
## [1] "1.3, 4.4, setosa"
## [1] "1.6, 5, setosa"
## [1] "1.9, 5.1, setosa"
## [1] "1.4, 4.8, setosa"
## [1] "1.6, 5.1, setosa"
## [1] "1.4, 4.6, setosa"
## [1] "1.5, 5.3, setosa"
## [1] "1.4, 5, setosa"
## [1] "4.7, 7, versicolor"
## [1] "4.5, 6.4, versicolor"
## [1] "4.9, 6.9, versicolor"
## [1] "4, 5.5, versicolor"
## [1] "4.6, 6.5, versicolor"
## [1] "4.5, 5.7, versicolor"
## [1] "4.7, 6.3, versicolor"
## [1] "3.3, 4.9, versicolor"
## [1] "4.6, 6.6, versicolor"
## [1] "3.9, 5.2, versicolor"
## [1] "3.5, 5, versicolor"
## [1] "4.2, 5.9, versicolor"
## [1] "4, 6, versicolor"
## [1] "4.7, 6.1, versicolor"
## [1] "3.6, 5.6, versicolor"
```

```
## [1] "4.4, 6.7, versicolor"
## [1] "4.5, 5.6, versicolor"
## [1] "4.1, 5.8, versicolor"
## [1] "4.5, 6.2, versicolor"
## [1] "3.9, 5.6, versicolor"
## [1] "4.8, 5.9, versicolor"
## [1] "4, 6.1, versicolor"
## [1] "4.9, 6.3, versicolor"
## [1] "4.7, 6.1, versicolor"
## [1] "4.3, 6.4, versicolor"
## [1] "4.4, 6.6, versicolor"
## [1] "4.8, 6.8, versicolor"
## [1] "5, 6.7, versicolor"
## [1] "4.5, 6, versicolor"
## [1] "3.5, 5.7, versicolor"
## [1] "3.8, 5.5, versicolor"
## [1] "3.7, 5.5, versicolor"
## [1] "3.9, 5.8, versicolor"
## [1] "5.1, 6, versicolor"
## [1] "4.5, 5.4, versicolor"
## [1] "4.5, 6, versicolor"
## [1] "4.7, 6.7, versicolor"
## [1] "4.4, 6.3, versicolor"
## [1] "4.1, 5.6, versicolor"
## [1] "4, 5.5, versicolor"
## [1] "4.4, 5.5, versicolor"
## [1] "4.6, 6.1, versicolor"
## [1] "4, 5.8, versicolor"
## [1] "3.3, 5, versicolor"
## [1] "4.2, 5.6, versicolor"
## [1] "4.2, 5.7, versicolor"
## [1] "4.2, 5.7, versicolor"
## [1] "4.3, 6.2, versicolor"
## [1] "3, 5.1, versicolor"
## [1] "4.1, 5.7, versicolor"
## [1] "6, 6.3, virginica"
## [1] "5.1, 5.8, virginica"
## [1] "5.9, 7.1, virginica"
## [1] "5.6, 6.3, virginica"
## [1] "5.8, 6.5, virginica"
## [1] "6.6, 7.6, virginica"
## [1] "4.5, 4.9, virginica"
## [1] "6.3, 7.3, virginica"
## [1] "5.8, 6.7, virginica"
## [1] "6.1, 7.2, virginica"
## [1] "5.1, 6.5, virginica"
## [1] "5.3, 6.4, virginica"
## [1] "5.5, 6.8, virginica"
## [1] "5, 5.7, virginica"
## [1] "5.1, 5.8, virginica"
## [1] "5.3, 6.4, virginica"
## [1] "5.5, 6.5, virginica"
## [1] "6.7, 7.7, virginica"
## [1] "6.9, 7.7, virginica"
```

```
## [1] "5, 6, virginica"
## [1] "5.7, 6.9, virginica"
## [1] "4.9, 5.6, virginica"
## [1] "6.7, 7.7, virginica"
## [1] "4.9, 6.3, virginica"
## [1] "5.7, 6.7, virginica"
## [1] "6, 7.2, virginica"
## [1] "4.8, 6.2, virginica"
## [1] "4.9, 6.1, virginica"
## [1] "5.6, 6.4, virginica"
## [1] "5.8, 7.2, virginica"
## [1] "6.1, 7.4, virginica"
## [1] "6.4, 7.9, virginica"
## [1] "5.6, 6.4, virginica"
## [1] "5.1, 6.3, virginica"
## [1] "5.6, 6.1, virginica"
## [1] "6.1, 7.7, virginica"
## [1] "5.6, 6.3, virginica"
## [1] "5.5, 6.4, virginica"
## [1] "4.8, 6, virginica"
## [1] "5.4, 6.9, virginica"
## [1] "5.6, 6.7, virginica"
## [1] "5.1, 6.9, virginica"
## [1] "5.1, 5.8, virginica"
## [1] "5.9, 6.8, virginica"
## [1] "5.7, 6.7, virginica"
## [1] "5.2, 6.7, virginica"
## [1] "5, 6.3, virginica"
## [1] "5.2, 6.5, virginica"
## [1] "5.4, 6.2, virginica"
## [1] "5.1, 5.9, virginica"
```

A função `sum` pode também ser útil. Lembre-se de usar o comando `?` para ajuda.

```r
## RESOLUÇÃO DO EXERCÍCIO 04

knn_func_iris <- function(k, x, D) {
    valor_predito <- D %>%
    # calculando as distancias Euclideanas para cada ponto em D
    mutate(dist = sqrt((x[1] - x_1)^2 + (x[2] - x_2)^2)) %>%
    # ordenando os resuldados obtidos em ordem crescente
    arrange(dist) %>%
    # separando os rótulos dos k pontos mais próximos
    slice(1:k) %>%
    # contando a quantidade de cada rótulo obtido para k vizinhos
    count(y) %>%
    # identificando qual dos rótulos foi mais frequente
    slice_max(n, with_ties = FALSE) %>%
    pull(y)
  return(valor_predito)
}


library(tidyverse)
```

```r
data("iris") # Carrega o banco no ambiente global
iris <- as_tibble(iris) %>%  # Converte para a dataframe tibble
  select(Petal.Length,Sepal.Length,Species) %>% # Seleciona colunas da dataframe
  rename( x_1 = Petal.Length, x_2 = Sepal.Length, y = Species) # Renomeia as colunas


## TESTANDO A CLASSIFICACAO

k_10 <- 10
k_1 <- 1
D <- iris


# TESTE K=10
predicao_k10 <- iris %>%
  select(x_1, x_2) %>%
  rowwise() %>%
  mutate(prediction = knn_func_iris(k_10, c(x_1, x_2), D = iris))
  print(predicao_k10)
```

```
## # A tibble: 150 x 3
## # Rowwise:
##      x_1   x_2 prediction
##    <dbl> <dbl> <fct>
##  1   1.4   5.1 setosa
##  2   1.4   4.9 setosa
##  3   1.3   4.7 setosa
##  4   1.5   4.6 setosa
##  5   1.4   5   setosa
##  6   1.7   5.4 setosa
##  7   1.4   4.6 setosa
##  8   1.5   5   setosa
##  9   1.4   4.4 setosa
## 10   1.5   4.9 setosa
## # i 140 more rows
```

```r
# TESTE K=1
predicao_k1 <- iris %>%
  select(x_1, x_2) %>%
  rowwise() %>%
  mutate(prediction = knn_func_iris(k_1, c(x_1, x_2), D = iris))
  print(predicao_k1)
```

```
## # A tibble: 150 x 3
## # Rowwise:
##      x_1   x_2 prediction
##    <dbl> <dbl> <fct>
##  1   1.4   5.1 setosa
##  2   1.4   4.9 setosa
##  3   1.3   4.7 setosa
##  4   1.5   4.6 setosa
##  5   1.4   5   setosa
```

```
##  6   1.7   5.4 setosa
##  7   1.4   4.6 setosa
##  8   1.5   5   setosa
##  9   1.4   4.4 setosa
## 10   1.5   4.9 setosa
## # i 140 more rows
```

Exercício 5 (opcional)

Em aula vimos como calcular a função de regressão $f : \mathcal{X} \to \mathcal{Y}$ ótima que minimiza o risco esperado:

$$\mathcal{R}(f) = \mathbb{E}_{XY}[\ell(Y, f(X))]$$

quando a função de perda é dada por $\ell_2(y, y') := (y - y')^2$. Essa função de perda é geralmente usada pela simplicidade de soluções. Mas existem outras funções de perda, como a função de perda do erro absoluto, que é dada por: $\ell_1(y, y') := |y - y'|$. Mostre que a função f ótima, que minimiza o risco esperado com essa função de perda, é dada por $f(x) := \text{Mediana}(Y|X = x)$. Suponha que a distribuição de Y é contínua.

dica 1: A mediana de uma variável aleatória contínua tomando valor em $\mathbb{R}$ é definida como sendo o valor real m tal que $P(Y > m) = P(Y < m) = 1/2$.

dica 2: A derivada de $\ell_1(y, y')$ em relação a y' quando $y' \neq y$ existe e é limitada. Nestes casos, sabe-se que

$$\frac{\partial}{\partial z}\mathbb{E}[\ell_1(Y, z)|X = x] = \mathbb{E}\Big[\frac{\partial}{\partial z}\ell_1(Y, z)\Big|X = x\Big].$$

Exercício 6 (opcional)

Considere que m pontos são espalhados uniformemente em uma hiperesfera de raio unitário e dimensão d. Mostre que a mediana da distância do ponto mais próximo à origem é dada por: $(1 - 0.5^{1/m})^{1/d}$.

Boa noite, professor. Conforme imagem, finalizei a tarefa mas não consegui submeter no GitHub antes de finalizar o prazo da atividade.