Lista 02 - Aprendizado de Máquinas

30 junho, 2022

Exercício 01

Usando a função: $x^2 - 2x^3 + 1$ gere pontos aleatórios a partir da relação $Y = f(x) + \varepsilon$, em que ε tem distribuição normal com média 0 e desvio padrão 0.5 (use a função `rnorm`). Para os valores de x, use pontos uniformemente amostrados em $[-1, 1]$ (use a função `runif`). Crie duas `tibble`, uma com 50 pares (x_i, y_i) para treinamento e outra com 100 pares para testes.

Faça a regressão dos dados gerados usando (i) uma regressão linear; (ii) o método kNN com $k = 1$; (iii) o método kNN com $k = 10$. Compare os erros nos 100 pares usados para teste. Comente os resultados em termos da relação de compromisso Viés-Variância (*bias-variance* tradeoff).

dicas:

Use o pacote `parsnip` que vem incluido no `tidymodels` para rodar o kNN e regressão linear. Assumindo que seu conjunto de treinamento está na `tibble` train e $k = 10$, teríamos;

```
library(tidymodels)

## Cria o modelo knn
knn.model <- nearest_neighbor(neighbors = 10,
                              weight_func = "rectangular",
                              dist_power = 2) %>%
             set_engine("kknn") %>%
             set_mode("regression")

## Faz o treinamento
knn.fit <- knn.model %>%
    fit( y ~ x, data = train)
```

A expressão `y ~ x` passada como primeiro argumento da função `fit` é conhecida como "fórmula" na linguagem `R`. Ela representa que queremos predizer a coluna y da tabela `train` a partir da coluna x.

Para fazer a predição, usaremos a função `predict` sobre a `tibble` test e depois mediremos a raiz do erro médio quadrático (RMSE) usando a biblioteca `yardstick` incluída no `tidymodels`.

```
## Faz a predição
test.pred <- knn.fit %>%
    predict( new_data = test) %>%
    bind_cols( test ) ## Adiciona as predicoes como coluna
                      ##  da tabela test

## Calcula o erro
rmse(test.pred, y, .pred)
```

Para rodar a regressão linear, teríamos o seguinte:

```
library(tidymodels)

## Cria o modelo linear
```

```
lin.model <- linear_reg() %>%
             set_engine("lm") %>%
             set_mode("regression")

## Faz o treinamento
lin.fit <- lin.model %>%
    fit( y ~ x, data = train)
```

A predição seria similar ao que foi feito acima para o kNN.

Exercício 02

Carregue o banco `PimaIndiansDiabetes` no R e faça um gráfico de pares usando `ggpairs` da biblioteca `GGally` (ou outra função/biblioteca equivalente). Descreva o que pode ser visto neste gráfico.

dicas:

Você pode carregar o banco no R usando os comandos:

```
library(tidyverse)
library(mlbench)
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)
```

A biblioteca `mlbench` contém vários bancos utilizados na literatura de aprendizado de máquinas. Use o comando `?PimaIndiansDiabetes` para saber mais sobre esse banco.

Exercício 03

Use o método kNN no banco `PimaIndiansDiabetes` variando os valores de k. Use o método de validação por conjunto de testes e escolha o melhor k de acordo com o critério da taxa de erro de classificação.

dicas:

Usaremos a função `initial_split` da biblioteca `rsample` do `tidymodels` para particionar os nossos dados. Como ilustrado abaixo:

```
library(tidymodels)
tt.split <- initial_split(pima_indians,prop = 0.8)

# Train/test serão nossos conjunto de treinamento e testes
train <- training(tt.split)
test  <- testing(tt.split)
```

Para fazer o treinamento, podemos fazer como no exercício anterior, mas alterando o modo de operação do kNN para "classificação" e a fórmula para `diabetes ~ .` (ou seja, predição de diabetes usando todos os preditores - o ponto do lado direito do til representa todos os preditores). Além disso, conseguimos calcular o erro de classificação usando a função `accuracy` no lugar do `rmse` do exercício anterior (acurácia é definida como $1 - t_{err}$, em que t_{err} é a taxa de erro).

Exercício 04

Use o método kNN no banco `PimaIndiansDiabetes` variando os valores de k. Use o método de validação cruzada para escolher o melhor k de acordo com o critério da taxa de erro de classificação. Faça um gráfico do erro de classificação contra $1/k$ (use uma quantidade de pontos k adequada para o computador que está rodando, um exemplo seria variar de 1 à 20 pulando de 2 em 2, ou de 3 em 3, etc.).

dicas:

Para gerar os conjuntos da validação cruzada, use o método `vfold` do pacote `rsample` contido no `tidymodels`.

```
library(tidymodels)

# Gera conjuntos para validação cruzada com 10 folds
#  v é o número de folds. (Você deve fazer isso
#  somente uma vez para todos os $k$)
cv.split <- vfold_cv(pima_indians,v=10)
```

Essa função gera os subconjuntos de validação cruzada que podemos usar para testar nosso algoritmo.

Após isso, usamos a função `fit_resamples` para calcular o erro de validação cruzada para uma escolha de parâmetro k (nesse caso $k = 20$).

```
## Montamos o nosso modelo de kNN
knn.model <- nearest_neighbor(neighbors = 20,              # $k$ = número de vizinhos
                              weight_func = "rectangular", # ponderação retangular
                              dist_power = 2) %>%          # distância Euclideana
             set_engine("kknn") %>%
             set_mode("classification")

# Calculando a acurraria usando os conjuntos de
#  validação cruzada. A função fit_resamples faz
#  o trabalho sujo para você aqui.
knn.fits <- fit_resamples(knn.model,    # Modelo
              diabetes ~ .,             # Fórmula (predizer diabetes com todos preditores)
              resamples = cv.split)     # Folds

# Você pode coletar a acurácia chamando a função collect_metrics
knn.fits %>% collect_metrics()
```

.metric	.estimator	mean	n	std_err	.config
accuracy	binary	0.7577751	10	0.0136731	Preprocessor1_Model1
roc_auc	binary	0.8120685	10	0.0166403	Preprocessor1_Model1

Você precisa repetir o processo de definir o modelo e chamar `fit_resamples` para cada parâmetro k que você deseja testar. Note que você precisa calcular o conjunto de validação cruzada chamando `vfold_cv` apenas uma vez.

Exercício 05 (Opcional)

Repita o Exercício 01 usando a função: $f(x) = 2x + 1$. Comente os seus resultados em relação ao que foi obtido no Exercício 01.

Exercício 06 (Opcional)

Mostre passo a passo como obter a decomposição viés-variância do erro de regressão com função de perda quadrática.