Lista 3 - Tópicos em ML

Luben M. C. Cabezas Reinaldo C. Anacleto

Primeiramente, importando bibliotecas que serão utilizadas:

```r
# bibliotecas do R
library(reticulate)
library(ggplot2)
library(MASS)
library(BART)
library(purrr)
```

```python
# pacotes para transformacao dos dados e graficos
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# pre-processamento de dados
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# modulo para acompanhar progresso
from tqdm import tqdm
```

Exercício 1

Importando o conjunto de dados de proteinas:

```python
# importando os dados
protein_data = pd.read_csv("data/CASP.csv")
# visualizando numero de variaveis, observações e tipo de cada variavel
```

```
protein_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45730 entries, 0 to 45729
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   RMSD    45730 non-null  float64
 1   F1      45730 non-null  float64
 2   F2      45730 non-null  float64
 3   F3      45730 non-null  float64
 4   F4      45730 non-null  float64
 5   F5      45730 non-null  float64
 6   F6      45730 non-null  float64
 7   F7      45730 non-null  float64
 8   F8      45730 non-null  int64  
 9   F9      45730 non-null  float64
dtypes: float64(9), int64(1)
memory usage: 3.5 MB
```

Dividindo o conjunto em treino, calibração e teste. O conjunto de calibração será usado apenas para os métodos conformal:

```
# Dividindo em treino e teste
train_val_data, test_data = train_test_split(
    protein_data, test_size=0.1, random_state=42
)

# Dividindo em treino e calibracao
train_data, calib_data = train_test_split(
    train_val_data, test_size=0.5, random_state=125
)

# Transformando em arrays
# arrays para conformal
X_train, y_train = train_data.iloc[:, 1:].values, train_data.iloc[:, 0].values
X_calib, y_calib = calib_data.iloc[:, 1:].values, calib_data.iloc[:, 0].values
X_test, y_test = test_data.iloc[:, 1:].values, test_data.iloc[:, 0].values
# array para bayesiano
X_train_all, y_train_all = train_val_data.iloc[:, 1:], train_val_data.iloc[:, 0]
```

Item 1

Item 2

Item 3

Item 4

Para esse exemplo, utilizaremos o conjunto de dados de treino completo. Utilizaremos o pacote BART no R, usando o reticulate como forma de traduzir os objetos em python para objetos em R. Temos os conjuntos de treino e teste:

```
# transformando objetos usando o reticulate
X_train_b <- py$X_train_all
y_train_b <- py$y_train_all
X_test_b <- py$X_test
y_test_b <- py$y_test
```

Agora ajustando o BART:

```
# Set seed for reproducibility
set.seed(686)
# Fit BART model
post <- wbart(X_train_b, y_train_b, X_test_b, ndpost = 1000)
```

```
*****Into main of wbart
*****Data:
data:n,p,np: 41157, 9, 4573
y1,yn: -6.701112, -5.957112
x1,x[n*p]: 5302.980000, 37.945800
xp1,xp[np*p]: 7031.440000, 22.507100
*****Number of Trees: 200
*****Number of Cut Points: 100 ... 100
*****burn and ndpost: 100, 1000
*****Prior:beta,alpha,tau,nu,lambda: 2.000000,0.950000,0.371213,3.000000,5.224211
*****sigma: 5.178756
*****w (weights): 1.000000 ... 1.000000
*****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,9,0
*****nkeeptrain,nkeeptest,nkeeptestme,nkeeptreedraws: 1000,1000,1000,1000
*****printevery: 100
*****skiptr,skipte,skipteme,skiptreedraws: 1,1,1,1
```

```
MCMC
done 0 (out of 1100)
done 100 (out of 1100)
done 200 (out of 1100)
done 300 (out of 1100)
done 400 (out of 1100)
done 500 (out of 1100)
done 600 (out of 1100)
done 700 (out of 1100)
done 800 (out of 1100)
done 900 (out of 1100)
done 1000 (out of 1100)
time: 279s
check counts
trcnt,tecnt,temecnt,treedrawscnt: 1000,1000,1000,1000
```

Obtemos agora uma região preditiva simétrica e a região quantílica:

```r
alpha <- 0.05
# Obtendo os lower e upper bounds da regiao simetrica para o conjunto de teste
means <- post$yhat.test.mean
std_dev <- sqrt(mean(post$sigma)^2 + apply(post$yhat.test, 2, sd)^2)
lower_bound_sym <- means - 1.96 * std_dev
upper_bound_sym <- means + 1.96 * std_dev

sigmas <- post$sigma[101:1100]
# Obtendo os lower e upper bounds da região baseada nos quantis
# encontrando quantil inferior usando monte carlo
y_new <- 1:length(y_test_b) |>
map(function(.x){
  y_sim <- post$yhat.test[, .x]
  return(
    rnorm(length(y_sim), mean = y_sim, sd = sigmas)
  )
}) |> unlist() |> matrix(nrow = 1000)

# grid em y
lower_bound_q <- apply(y_new, 2, quantile, probs = alpha/2)
upper_bound_q <- apply(y_new, 2, quantile, probs = (1 - alpha/2))
```

Tendo ambas as regiões, podemos a seguir calcular a cobertura empírica nesses casos:

```r
cover_sym <- ((lower_bound_sym <= y_test_b) &
(upper_bound_sym >= y_test_b)) |>
mean()

cover_q <- ((lower_bound_q <= y_test_b) &
(upper_bound_q >= y_test_b)) |>
mean()

data.frame("Região Preditiva" = c("Simétrica", "Quantílica"),
"Cobertura empírica" = c(cover_sym, cover_q) |> round(4))
```

Região.Preditiva	Cobertura.empírica
Simétrica	0.9495
Quantílica	0.9473

Percebe-se que ambas regiões são razoavelmente próximas da cobertura nominal 0.95, tendo porém uma leve sub-cobertura, principalmente a região quantílica que está um pouco mais distante do nível nominal que o intervalo simétrico. Podemos também observar pela Figura 1 as regiões preditivas estimadas no conjunto de teste para 5 diferentes valores.

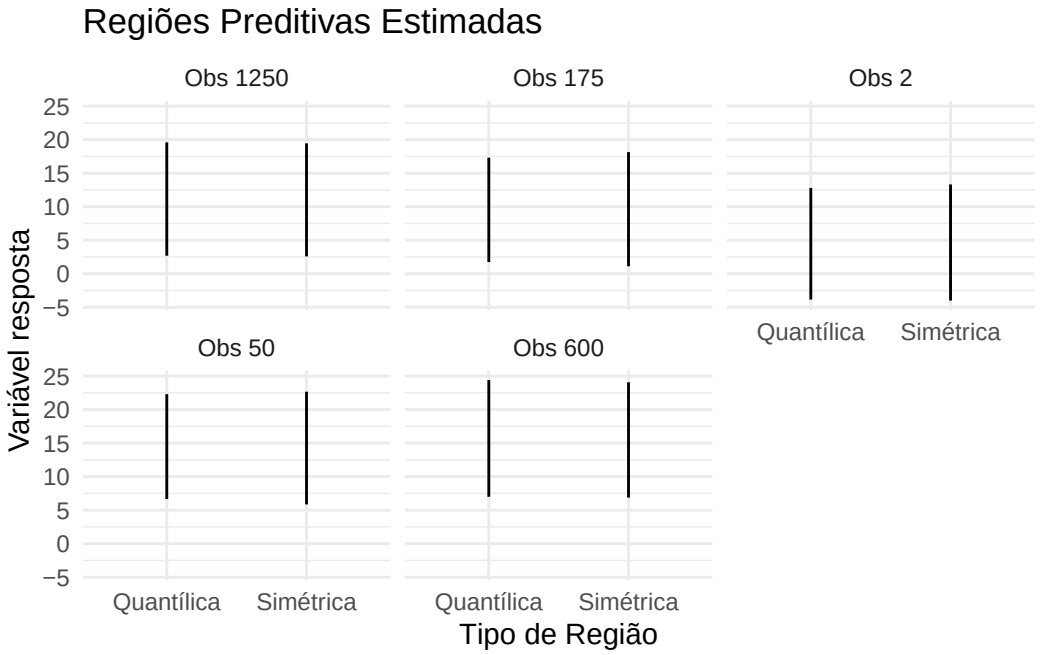

Figure 1: Intervalos preditivos bayesianos para as observações selecionadas

Percebemos que há poucas diferenças entre os tipos de regiões, com ambas tendo tamanhos similares e razoavelmente largos.