Lista 1 - Tópicos em ML

Luben M. C. Cabezas Reinaldo C. Anacleto

Primeiramente, importando bibliotecas que serão utilizadas

```
# pacotes para transformacao dos dados e graficos
library(dplyr)
library(tidyr)
library(ggplot2)
library(rsample)

# pacotes para o lasso, arvore de decisao e floresta aleatoria
library(glmnet)
library(rpart)
library(rpart.plot)
# random forest com criterio poisson
library(rfCountData)

# pacotes para redes neurais
library(torch)
library(luz)
library(torchdatasets)
```

Exercício 1

Importando os dados e visualizando uma descrição geral dos dados através da função *str()* do R:

```
data_list <- read.csv("data/time.csv") |> mutate_if(is.character,as.factor)
data_list |> str()
```

```
'data.frame':   1000 obs. of  12 variables:
 $ age          : int  56 46 32 60 25 38 56 36 40 28 ...
 $ gender       : Factor w/ 3 levels "female","male",..: 2 1 2 3 2 2 2 2 3 3 ...
 $ time_spent   : int  3 2 8 5 1 3 8 4 7 2 ...
 $ platform     : Factor w/ 3 levels "Facebook","Instagram",..: 2 1 2 2 2 1 3 2 3 2 ...
 $ interests    : Factor w/ 3 levels "Lifestlye","Sports",..: 2 3 2 3 1 3 2 2 1 2 ...
 $ location     : Factor w/ 3 levels "Australia","United Kingdom",..: 2 2 1 2 1 3 3 1 1 1 ...
 $ demographics: Factor w/ 3 levels "Rural","Sub_Urban",..: 3 3 2 3 3 3 3 3 2 2 ...
 $ profession   : Factor w/ 3 levels "Marketer Manager",..: 2 3 1 3 2 1 3 1 1 1 ...
 $ income       : int  19774 10564 13258 12500 14566 19179 16881 13636 16030 10223 ...
 $ indebt       : Factor w/ 2 levels "False","True": 2 2 1 1 1 2 2 2 1 2 ...
 $ isHomeOwner : Factor w/ 2 levels "False","True": 1 2 1 2 2 2 2 1 1 1 ...
 $ Owns_Car     : Factor w/ 2 levels "False","True": 1 2 1 1 2 2 2 2 2 2 ...
```

Nesse banco de dados temos um total de 1000 observações e 12 variáveis no total, com a variável resposta sendo quantitativa discreta (valores inteiros). Além disso, entre as 11 covariáveis, 9 são categóricas e as 2 outras são quantitativas. Entre as categóricas, pode-se visualizar a frequência de cada categoria em cada covariavel na Figura 1.

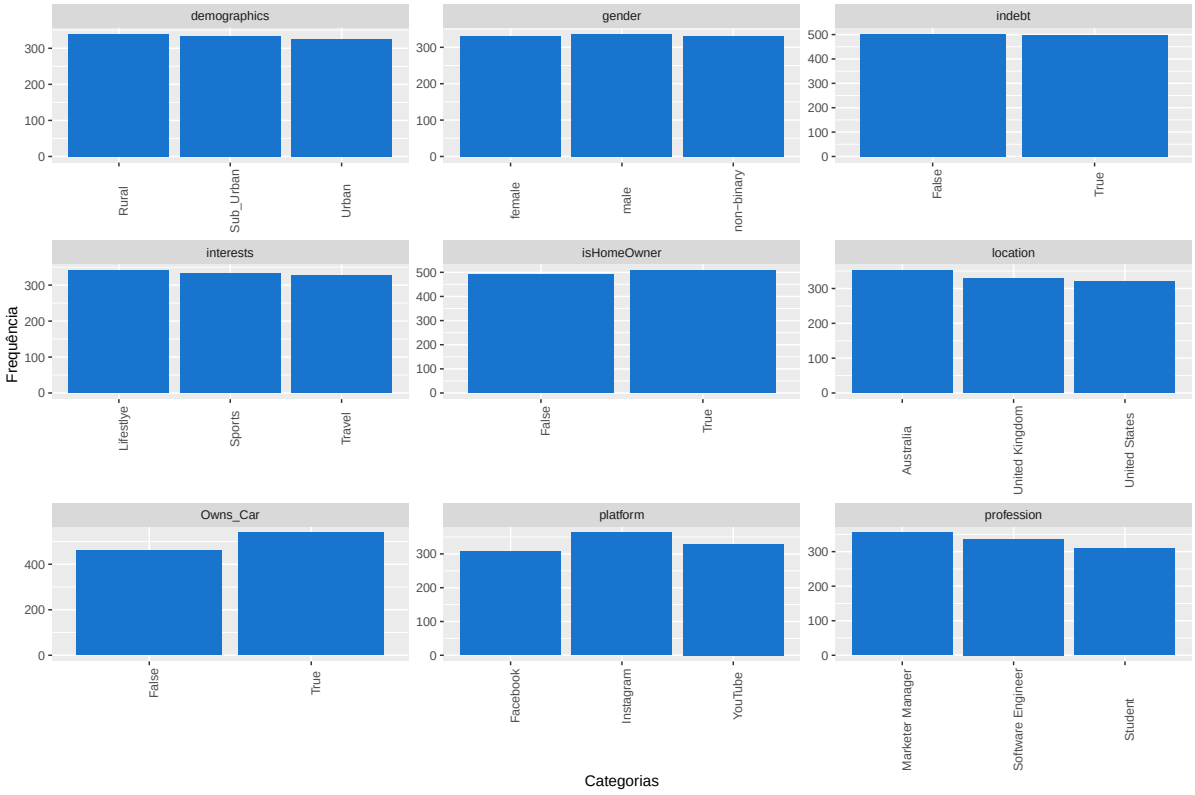

Figure 1: Frequência de cada categoria em cada covariavel categórica

Pela Figura 1, visualizamos que para cada variável categórica, as frequências de cada categoria são bem equilibradas e igualmente distribuidas. Na Figura 2 podemos visualizar também os histogramas das duas covariáveis quantitativas:

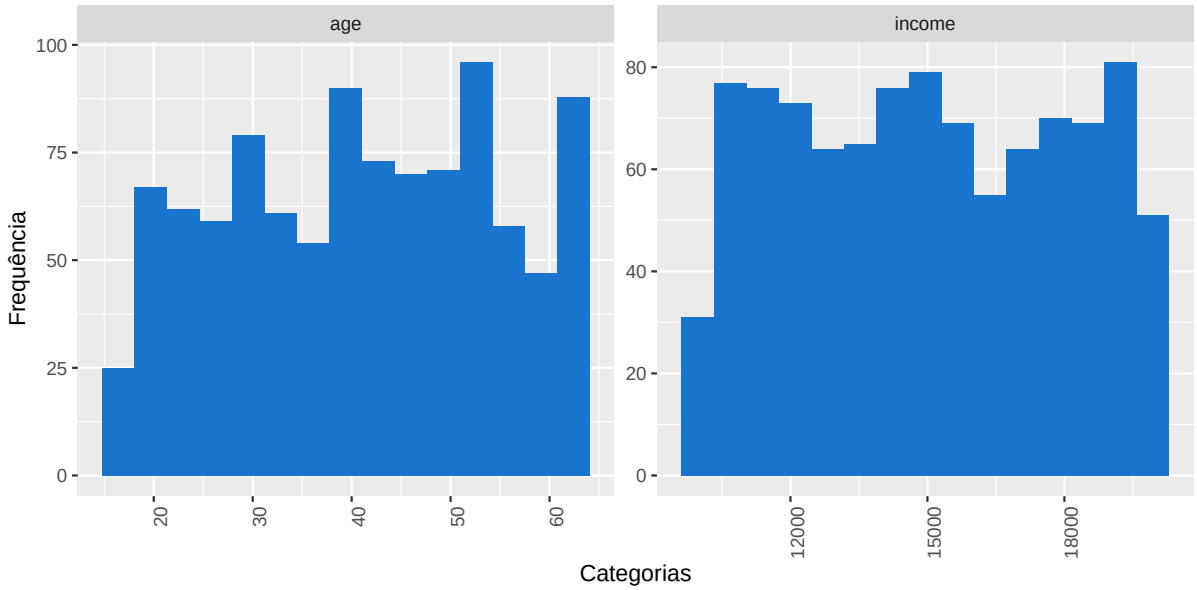

Figure 2: Histograma das duas covariáveis quantitativas

Em geral, pela Figura 2 nota-se um equilibrio nas distribuições da idade e renda, com tais distribuições sendo muito similares a Uniforme. Enfim, podemos visualizar a variável resposta associada ao tempo gasto por cada usuário nas redes sociais através da Figura 3.

Percebemos que o tempo gasto por cada usuário se distribui de forma razoavelmente uniforme entre 1 a 9 horas por dia, tendo então, cada contagem de horas gastas na rede social aproximadamente equiprováveis.

Nesse contexto, em que temos 1000 observações, consideramos plausivel utilizar 70% da amostra para treino (700 observações) e 30% para teste (300 observações), garantindo assim, que estamos utilizando o máximo de amostras possíveis para o treinamento dos modelos ao passo que conseguimos estimar o risco de forma consistente.

```
set.seed(145, sample.kind = "Rounding")
# dividindo conjunto de dados usando initial_split do rsample
# obtendo configuração do splitting
data_split <- data_list |> initial_split(0.7)

# dividindo o conjunto em treino e teste
train_df <- training(data_split)
```

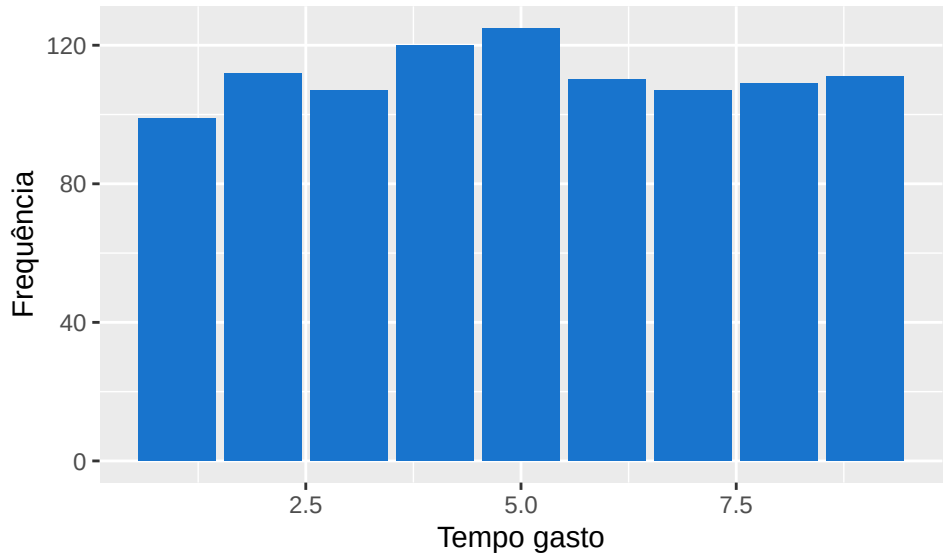

Figure 3: Histograma do tempo gasto por cada usuário na rede social

```
test_df  <- testing(data_split)
```

Assim, vamos agora ajustar o LASSO, árvore de decisão, floresta aleatória e 10 diferentes arquiteturas de redes neurais no conjunto de treino, e na sequência utilizar o conjunto de teste para comparar todos os métodos em termos do Erro Absoluto Médio, dado por:

$$\hat{R}(g) = \frac{1}{|I_{\text{teste}}|} \sum_{i \in I_{\text{teste}}} |g(x_i) - y_i| \,, \tag{1}$$

tal que I_{teste} é o conjunto dos indices pertencentes a amostra de teste. Para a comparação dos erros absolutos médios, serão utilizados intervalos de confiança assintóticos com 95% de confiança dos erros. Como a variável resposta possui valores quantitativos discretos, adotaremos como perda/critério principal para o ajuste de todos os métodos a seguir o resíduo deviance poisson. Adicionalmente, compararemos cada método ajustado via critério de poisson com o critério da perda quadrática via erro absoluto médio no conjunto de teste.

Ajustando o LASSO

Para o ajuste do LASSO, utilizaremos o pacote *glmnet*. Primeiramente, otimizaremos o hiperparâmetro de regularização λ via K-fold com 5 folds fixos:

```r
# obtendo matriz de covariaveis para o ajuste
X_tr <- model.matrix(time_spent ~., data = train_df)[,-1]
y_tr <- train_df$time_spent

# matriz de covariavel do conjunto de teste
X_test <- model.matrix(time_spent ~ ., data = test_df)[,-1]
y_test <- test_df$time_spent

# obtendo o melhor parametro de regularizacao
# usando desvio da poisson
lasso_obj <- cv.glmnet(
    X_tr,
    y_tr,
    alpha = 1,
    intercept=TRUE,
    standardize = TRUE,
    family = "poisson",
    nfolds = 5)
```

O λ que minimiza o resíduo deviance poisson é de:

```r
lasso_obj$lambda.min
```

```
[1] 0.06587221
```

Tendo o gráfico do λ escolhido via validação cruzada dado pela Figura 4.

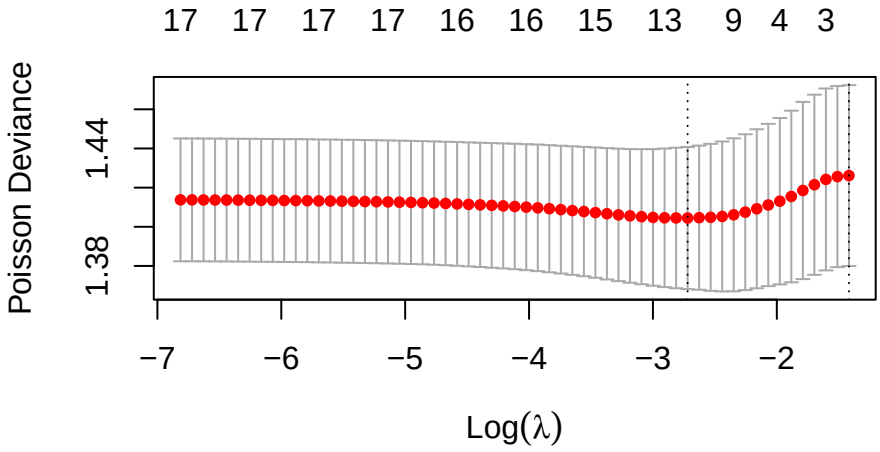

Figure 4: Resíduo deviance poisson médio para cada valor do parâmetro de regualização

Nota-se, pela parte superior do gráfico, que para o valor de λ que minimiza o resíduo deviance, cerca de 13 covariáveis juntamente com o intercepto não serão nulos, tendo a seguir os coeficientes estimados pelo LASSO:

```
coef(lasso_obj, s = lasso_obj$lambda.min)
```

```
18 x 1 sparse Matrix of class "dgCMatrix"
                                      s1
(Intercept)                1.7433895399
age                       -0.0002177221
gendermale                -0.0687950363
gendernon-binary           .
platformInstagram          .
platformYouTube           -0.0580565009
interestsSports           -0.0830509591
interestsTravel            .
locationUnited Kingdom    -0.0390037801
locationUnited States     -0.0383962161
demographicsSub_Urban      0.0056448563
demographicsUrban         -0.0801158711
professionSoftware Engineer -0.0061784169
professionStudent          0.0117052162
income                     .
indebtTrue                 .
isHomeOwnerTrue            0.0120600302
Owns_CarTrue              -0.0245372477
```

Percebe-se que as covariáveis com coeficientes não nulos em geral possuem valores razoavelmente baixos, indicando que mesmo que selecionadas, ainda assim, essas covariáveis são pouco significativas para explicar o tempo gasto do usuário. Além disso, nota-se também que apenas as covariáveis *income*, *indebt*, o fator Travel de *interests*, a plafatorma do instagram e o genero não binário, não foram selecionadas. Podemos comparar o erro absoluto médio do o LASSO ajustado via perda quadrática e Poisson.

```
# ajustando o lasso
lasso_obj_quad <- cv.glmnet(
    X_tr,
    y_tr,
    alpha = 1,
    intercept=TRUE,
    standardize = TRUE,
```

```
    family = "gaussian",
    nfolds = 5)

# computando as predicoes no teste
pred_lasso_pois <- predict(lasso_obj,
s = lasso_obj$lambda.min,
newx = X_test, type = "response")

pred_lasso_quad <- predict(lasso_obj_quad,
s = lasso_obj_quad$lambda.min,
newx = X_test, type = "response")

# computando o erro absoluto medio
MAE_lasso_pois <- mean(abs(pred_lasso_pois - y_test))
MAE_lasso_quad <- mean(abs(pred_lasso_quad - y_test))

# comparando via tabela
data.frame("Método" = c("Lasso Poisson", "Lasso Quadrático"),
           "EAM" = c(MAE_lasso_pois, MAE_lasso_quad))
```

Método	EAM
Lasso Poisson	2.176936
Lasso Quadrático	2.180921

Nota-se pouca diferença entre as abordagens, tendo o Lasso Poisson uma pequena melhora no erro absoluto médio comparado ao Lasso quadrático.

Árvore de regressão

Vamos a seguir ajustar um árvore de regressão, utilizando como perda para o particionamento o resíduo deviance da poisson:

```
# ajustando arvore de regressao
tree_fit <- rpart(time_spent ~ ., method = "poisson", data = train_df)

# encontrando melhor CCP-alpha para poda de
# acordo com validacao cruzada
best_cp <- tree_fit$cptable[which.min(tree_fit$cptable[, "xerror"]),
"CP"]
```

A seguir, pela Figura 5 podemos visualizar a árvore com os parâmetros default do pacote *rpart* que inclui uma poda padrão por custo de complexidade fixo em 0.01:

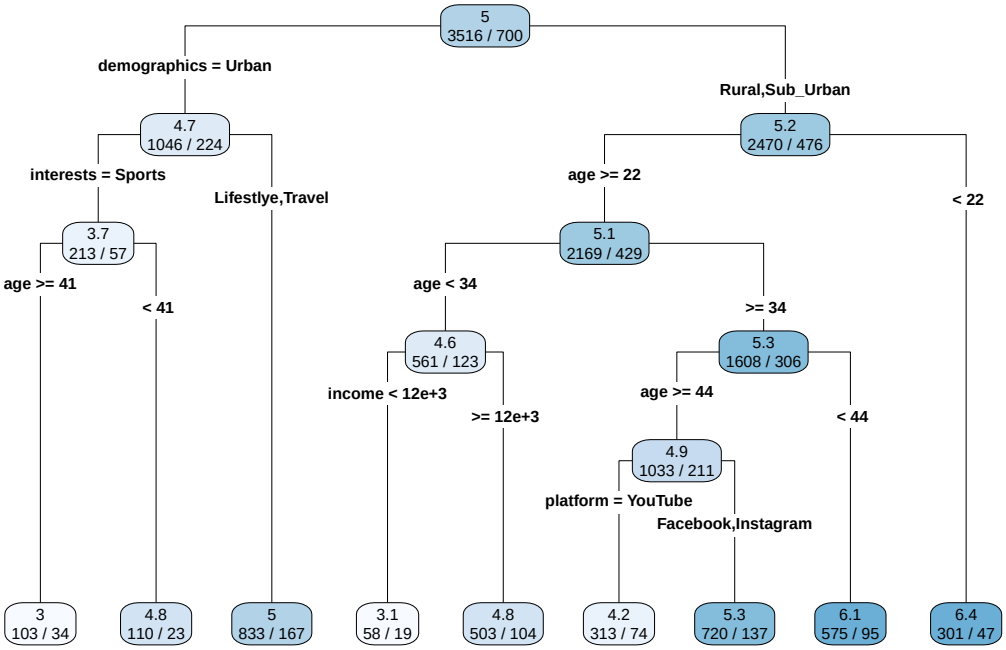

Figure 5: Árvore de regressão com poda padrão

Percebe-se que o ajuste do método com poda padrão nos dá uma árvore interpretável, com as divisões sendo principalmente associadas a demografia, interesse, idade, plataforma e salário, tendo a idade como critério de divisão em diversos segmentos da árvore de regressão. A seguir, podemos ainda obter a árvore podada com o custo de complexidade com a menor perda possível calculada via validação cruzada:

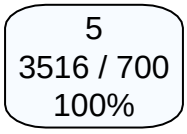

Figure 6: Árvore de regressão com a melhor poda

Pela Figura 6, percebe-se que a árvore podada com o mais ótimo custo de complexidade é uma árvore trivial, com apenas uma folha. Isso implica que as possíveis divisões obtidas anteriormente na verdade não possuem ganho de informação justificavel para sua inclusão e podem ser descartadas em função de reduzir a complexidade da árvore de regressão. Ou seja, as covariáveis em geral não possuem grande influência sobre a variável resposta. Podemos ao final comparar a árvore com poda ótima, a árvore com poda padrão e a árvore com poda ótima e critério quadrático através do erro absoluto médio computado no teste:

```r
tree_quad_fit <- rpart(time_spent ~ ., method = "anova", data = train_df)

# encontrando melhor CCP-alpha para poda de
# acordo com validacao cruzada
best_cp_quad <- tree_quad_fit$cptable[which.min(tree_quad_fit$cptable[, "xerror"]),
"CP"]

p_tree_quad_fit <- prune(tree_quad_fit, cp = best_cp_quad)

# predicoes no teste
pred_tree_pois <- predict(tree_fit, test_df)
pred_tree_pois_prune <- predict(p_tree_fit, test_df)
pred_tree_quad_prune <- predict(p_tree_quad_fit, test_df)

# computando MAE
MAE_tree_pois <- mean(abs(pred_tree_pois - y_test))
MAE_tree_pois_prune <- mean(abs(pred_tree_pois_prune - y_test))
MAE_tree_quad <- mean(abs(pred_tree_quad_prune - y_test))

# comparando via tabela
data.frame("Método" = c("Árvore Poisson", "Árvore Poisson poda ótima",
"Árvore quadrática poda ótima"),
           "EAM" = c(MAE_tree_pois, MAE_tree_pois_prune, MAE_tree_quad))
```

Método	EAM
Árvore Poisson	2.268874
Árvore Poisson poda ótima	2.133200
Árvore quadrática poda ótima	2.133200

Percebe-se que, a árvore poisson com poda ótima, juntamente com a árvore quadrática com poda ótima, tem a menor risco comparada a árvore poisson com perda padrão. Assim, conclui-se que a árvore trivial, que nos da como predição para todas observações a média amostral do tempo gasto nas redes sociais estimada no conjunto de treino, tem melhor poder preditivo que a árvore com poda padrão.

Florestas Aleatórias

A seguir, iremos ajustar uma floresta aleatória com critério poisson, utilizando-se $B = 500$ árvores e 100 de tamanho mínimo (número mínimo de amostras) por folha.

```
# ajustando arvore com criterio poisson
rf_fit <- rfPoisson(x = train_df |> select(-3),
y = train_df |> pull(3),
offset = rep(0, nrow(train_df)),
ntree = 500,
nodesize = 150)
```

Podemos visualizar a importância de cada covariável dada pela pureza das divisões associadas a cada uma através da Figura 7.

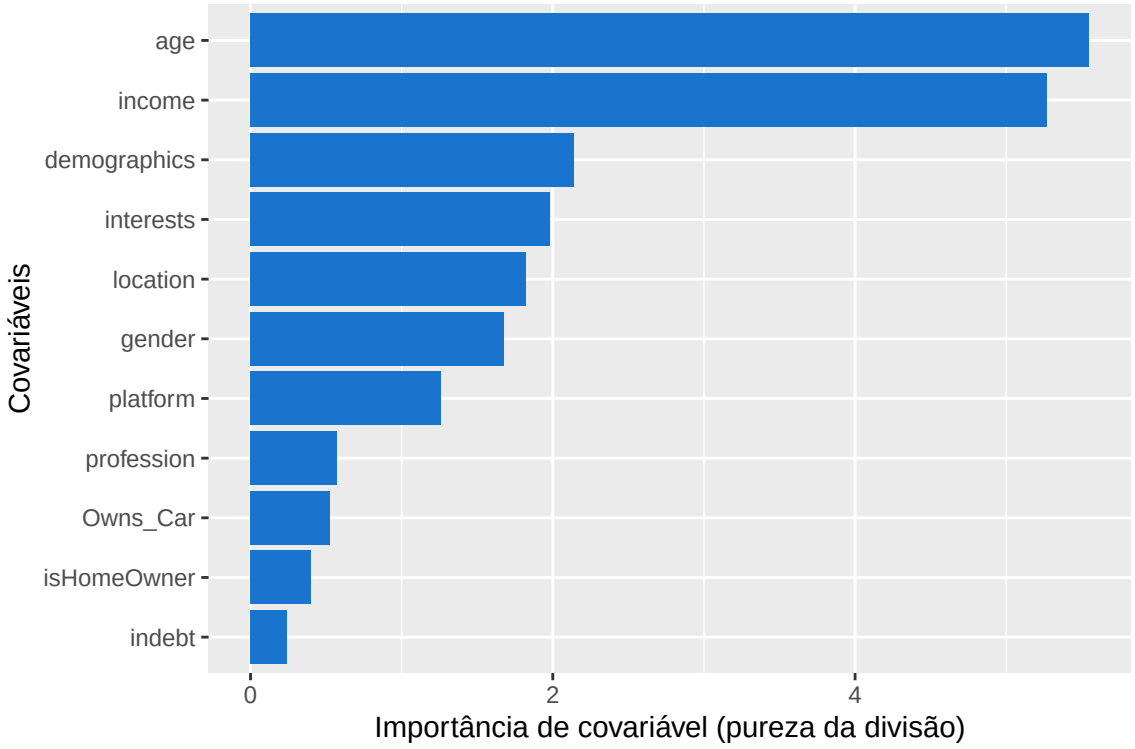

Figure 7: Importância de covariáveis na floresta

Observa-se pelo gráfico que, as covariáveis associadas a renda e idade tem maior importância, enquanto as covariáveis associadas a posse de carros e dívida tem menores importâncias. Já as demais covariáveis, tem importância média, tendo valores de 1 a 2 de pureza da partição. Podemos comparar essa floresta aleatória com a floresta aletória obtida usando-se como critério a perda quadrática, fixando-se o mesmo número de árvores e tamanho de folha que a floresta poisson:

```r
# ajustando modelo
rf_fit_quad <- randomForest::randomForest(
  x = train_df |> select(-3),
  y = train_df |> pull(3),
  ntree = 500,
  nodesize = 150)

# predizendo e computando erros absolutos medios
rf_pois_pred <- predict(rf_fit, offset = rep(0, nrow(test_df)) ,newdata = test_df |> selec
rf_quad_pred <- predict(rf_fit_quad, newdata = test_df |> select(-3))

MAE_rf_pois <- mean(abs(rf_pois_pred - y_test))
MAE_rf_quad <- mean(abs(rf_quad_pred - y_test))

data.frame("Método" = c("Floresta poisson", "Floresta quadrática"),
           "EAM" = c(MAE_rf_pois, MAE_rf_quad))
```

Método	EAM
Floresta poisson	2.145803
Floresta quadrática	2.158230

Visualiza-se pouca diferença de performance entre os métodos, com a floresta baseada no critério da perda poisson tendo um risco levemente menor que a floresta baseada na perda quadrática.

Redes Neurais

Para a criação e ajuste das redes neurais, faremos uso dos pacotes *torch*, *luz* e *torchdatasets*. Primeiramente, implementaremos uma função para a leitura dos dados no ambiente do *torch*:

```r
# criando funcao para ler os dados
hours_dataset <- dataset(

  name = "hours_dataset",

  initialize = function(X, y){

    # features
    self$X <- torch_tensor(X |> scale())
```

```
    self$y <- torch_tensor(y, dtype = torch_float())
  },

  .getitem = function(i) {
     list(X = self$X[i, ],
     y = self$y[i])

  },

  .length = function() {
    self$y$size()[[1]]
  }

)
```

Para o ajuste das redes neurais, dividiremos o conjunto de treino em treino e validação, utilizando a validação para avaliar a convergência e desempenho das redes neurais. Além disso, utilizaremos a função *dataloader* para dividir cada conjunto em minilotes, para assim se fazer uso do gradiente descendente estocástico:

```
# dividindo conjunto de treino em treino e validacao
# embaralhando indices
idx <- sample(1:nrow(X_tr), replace = FALSE)
idx_tr <- idx[1:500]
idx_val <- idx[501:nrow(X_tr)]

# numero de features
d_in <- ncol(X_tr)

# usando uma semente para reproducibilidade no torch
torch_manual_seed(145)

# indexando o data frame original
# conjunto de treino
train_dl <- hours_dataset(X_tr[idx_tr, ], y_tr[idx_tr]) |>
  dataloader(batch_size = 16, shuffle = TRUE)
valid_dl <- hours_dataset(X_tr[idx_val, ], y_tr[idx_val]) |>
  dataloader(batch_size = 16, shuffle = FALSE)
```

Agora, através da função *nn_module* criaremos as diversas arquiteturas de redes neurais. A configuração de cada arquitetura são detalhadas nos comentários do código a seguir:

```
# iniciando todas arquiteturas
# arq 1 Sem dropout, uma camada com 64 neuronios (64)
# funcao de ligacao relu
net_1 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 64),
      nn_relu(),
      nn_linear(64, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 2 Sem dropout, duas camadas, umas com 32 e outra com 18 e
# funcao de ligacao relu nas duas camadas
net_2 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 32),
      nn_relu(),
      nn_linear(32, 18),
      nn_relu(),
      nn_linear(18, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 3 Sem dropout, tres camadas, primeira com 48, segunda com
# 24 e ultima com 12
# funcao de ligacao relu na primeira camada para a segunda,
# funcao de ligacao selu da segunda para a terceira
# funcao de ligacao relu da terceira para a ultima
net_3 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
```

```
      nn_linear(d_in, 48),
      nn_relu(),
      nn_linear(48, 24),
      nn_selu(),
      nn_linear(24, 12),
      nn_relu(),
      nn_linear(12, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 4, 5 e 6: arq 1, 2 e 3 porem com dropout em todas camadas
net_4 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 64),
      nn_dropout(p = 0.5),
      nn_relu(),
      nn_linear(64, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

net_5 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 32),
      nn_dropout(p = 0.5),
      nn_relu(),
      nn_linear(32, 18),
      nn_dropout(p = 0.3),
      nn_relu(),
      nn_linear(18, 1)
    )
  },
```

```
  forward = function(x) {
    self$net(x)
  }
)

net_6 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 48),
      nn_dropout(p = 0.5),
      nn_relu(),
      nn_linear(48, 24),
      nn_dropout(p = 0.35),
      nn_selu(),
      nn_linear(24, 12),
      nn_dropout(p = 0.25),
      nn_relu(),
      nn_linear(12, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 7: mesmo que a arquitetura 6,
# porem usando silu, relu e selu ao final
net_7 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 48),
      nn_dropout(p = 0.5),
      nn_silu(),
      nn_linear(48, 24),
      nn_dropout(p = 0.35),
      nn_relu(),
      nn_linear(24, 12),
      nn_dropout(p = 0.25),
      nn_selu(),
      nn_linear(12, 1)
    )
```

```
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 8: mesmo que a arquitetura 5, porem usando a ligacao selu
# ao inves de relu
# e probabilidade de dropout igual a 0.5 nas duas camadas
net_8 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 32),
      nn_dropout(p = 0.5),
      nn_selu(),
      nn_linear(32, 18),
      nn_dropout(p = 0.5),
      nn_selu(),
      nn_linear(18, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# arq 9 e 10: mesmo que as arquiteturas 4 e 6 porém usando diferentes
# valores de learning rate, metodo de otimização e perda quadratica
# ao inves de poisson

# arquitetura igual a 4
net_9 <-  nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 64),
      nn_dropout(p = 0.5),
      nn_relu(),
      nn_linear(64, 1)
    )
  },
  forward = function(x) {
```

```
    self$net(x)
  }
)

# arquitetura igual a 6
net_10 <- net_6 <- nn_module(
  initialize = function(d_in) {
    self$net <- nn_sequential(
      nn_linear(d_in, 48),
      nn_dropout(p = 0.5),
      nn_relu(),
      nn_linear(48, 24),
      nn_dropout(p = 0.35),
      nn_selu(),
      nn_linear(24, 12),
      nn_dropout(p = 0.25),
      nn_selu(),
      nn_linear(12, 1)
    )
  },
  forward = function(x) {
    self$net(x)
  }
)

# formando lista com as arquiteturas
net_list <- list(net_1, net_2, net_3, net_4, net_5,
                 net_6, net_7, net_8, net_9, net_10)
```

Tendo todas as arquiteturas fixadas, consideraremos para todas as redes neurais um total de 150 épocas com early stopping em 20. Para as arquiteturas 1 a 8, utilizaremos o otimizador adam com taxa de aprendizado igual a 0.001 (default) e a perda poisson, enquanto que para as arquiteturas 9 e 10, usaremos o otimizador adadelta com taxa de aprendizado igual a 0.005 e perda quadrática. A seguir ajustamos todas as redes neurais e armazenamos seus objetos ajustados em uma lista:

```
fitted_list <- list()
i <- 1
for(net in net_list){
  if(i < 9){
    fitted <- net |>
```

```
  setup(
    loss = nn_poisson_nll_loss(),
    optimizer = optim_adam,
  ) |>
  set_hparams(d_in = d_in) |>
  set_opt_hparams(lr = 0.001) |>
  fit(
    train_dl,
    epochs = 150,
    valid_data = valid_dl,
    callbacks = list(
      luz_callback_early_stopping(
        patience = 15)
    ),
    verbose = FALSE)
  }else{
    fitted <- net |>
  setup(
    loss = nn_mse_loss(),
    optimizer = optim_adadelta,
  ) |>
  set_hparams(d_in = d_in) |>
  set_opt_hparams(lr = 0.005) |>
  fit(
    train_dl,
    epochs = 150,
    valid_data = valid_dl,
    callbacks = list(
      luz_callback_early_stopping(
        patience = 15)
    ),
    verbose = FALSE
    )
  }
  fitted_list[[i]] <- fitted
  i <- i + 1
}
```

Podemos então compara-las em termos do erro absoluto médio calculado no conjunto de teste a seguir:

```r
test_dl <- hours_dataset(X_test, y_test) |>
  dataloader(batch_size = 100, shuffle = FALSE)

# computando predicao para a lista de redes neurais ajustadas
# e em seguida computando risco de cada uma
i <- 1
mae_vec_net <- numeric(10)
net_name <- paste0("Rede ", 1:10)
for(fitted in fitted_list){
  preds <- fitted |>
  predict(test_dl) |> as.numeric()
  mae_vec_net[i] <- mean(abs(preds - y_test))
  i <- i + 1
}

data.frame("Configuração de rede neural" = net_name,
           "Erro Absoluto Médio" = mae_vec_net)
```

Configuração.de.rede.neural	Erro.Absoluto.Médio
Rede 1	3.531694
Rede 2	3.587239
Rede 3	3.544950
Rede 4	3.537390
Rede 5	3.570415
Rede 6	3.376723
Rede 7	3.524954
Rede 8	3.426747
Rede 9	3.587847
Rede 10	2.331186

Percebemos que, a última arquitetura (rede neural 10) tem um melhor desempenho em termos do erro absoluto médio, tendo um risco estimado em 2.33 enquanto todas as demais arquiteturas tem risco acima de 3. Logo, a rede neural baseada na perda quadrática parece ter um desempenho melhor que todas as redes baseadas na perda poisson. Para a comparação final de todos os métodos, apenas incluiremos a melhor arquitetura de rede neural selecionada.

```r
# salvando predicoes da melhor rede neural
pred_nnet_better <-  fitted_list[[10]] |>
predict(test_dl) |>
 as.numeric()
```

Comparação de todos os métodos

Para a comparação de cada método, utilizaremos o intervalo de confiança assintótico para o risco absoluto, fixando 95% de confiança. Sob a suposição de que as amostras do conjunto de teste são i.i.d, como $\hat{R}(g)$ dado pela Equação 1 é uma média amostral, vale pelo Teorema Central do Limite que:

$$\hat{R}(g) \approx \text{Normal}\left(R(g), \frac{1}{|I_{\text{teste}}|} \cdot \mathbb{V}[|g(X_1) - Y_1|]\right) . \tag{2}$$

Assim, estimando $\mathbb{V}[|g(X_1) - Y_1|]$ através da variância amostral $\hat{\sigma}^2$ dos desvios absolutos, obtemos o intervalo de confiança:

$$\hat{R}(g) \pm 2\sqrt{\frac{1}{|I_{\text{teste}}|}\hat{\sigma}^2} . \tag{3}$$

A seguir podemos calcular os IC's para os diferentes modelos ajustados até o momento, considerando-se apenas os melhores de cada classe de método na comparação:

```
# organizando metodos selecionados e suas predicoes
data_risks <- data.frame(
          `lasso` = as.numeric(abs(pred_lasso_pois - y_test)),
           `arvore de decisao` = abs(pred_tree_pois_prune - y_test),
           `floresta aleatória` = abs(rf_pois_pred - y_test),
           `rede neural` = abs(pred_nnet_better - y_test))

# pivotando os dados
conf_data <- data_risks |>
pivot_longer(cols = everything(), names_to = "metodos",
values_to = "perda") |>
group_by(metodos) |>
summarise(media = mean(perda),
          se = 2*sqrt(var(perda)/n()))
```

Podemos visualizar esse intervalos de confiança de 95% do risco pela Figura 8.

Nota-se visualmente que, entre os IC's, o lasso tem menor largura de intervalo, enquanto a arvore de decisão e floresta aleatoria tem IC's com menores valores. Já a rede neural, por ter um erro absoluto médio mais elevado, tem um IC com valores consideravelmente maiores que os demais. Apesar disso, todos os IC's em análise tem algum tipo de interseção, o que implica que não existe diferença significativa entre os desempenhos de cada método. Logo, os métodos desempenham de forma similar um ao outro. Isso pode ocorrer devido ao fato de as covariaveis consideradas no ajuste de todos os modelos não serem significativas para explicar a variável resposta. Apesar disso, o melhor modelo em termos de estimativa média (Árvore de Decisão) é dado pela média amostral da variável resposta no conjunto de treinamento, indicando que a variável resposta é independente das covariáveis consideradas.

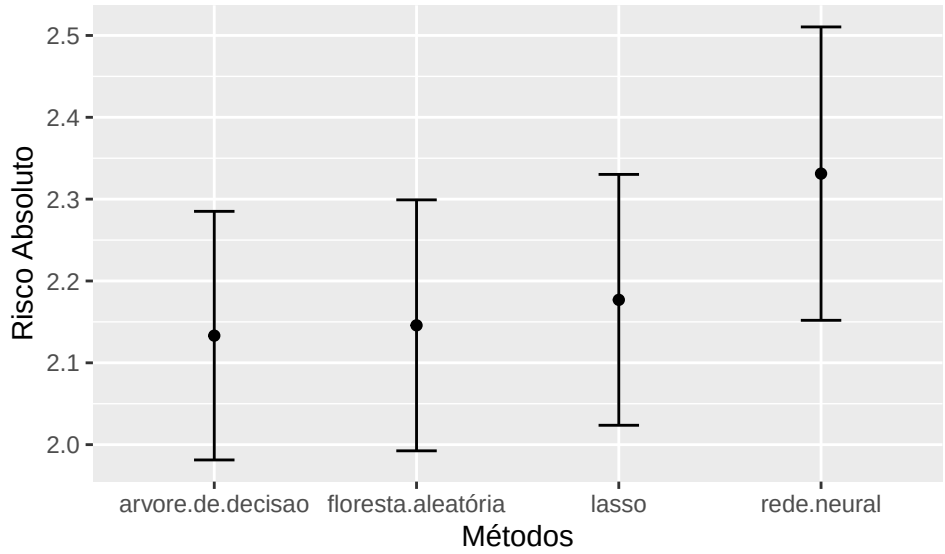

Figure 8: Intervalos de 95% de confiança para o risco de cada método.