Solução Lista 07

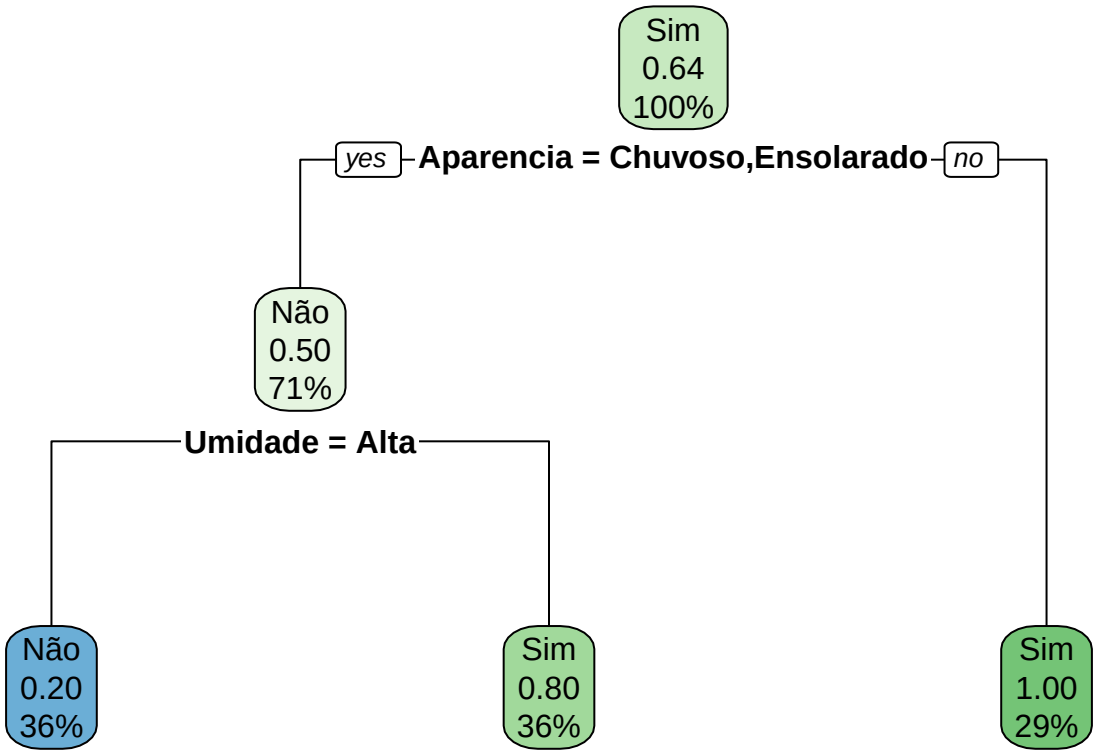
Nome: Vinicius de Oliveira Bezerra
E-mail: v.bezerra@aluno.ufabc.edu.br
Nome: Deyved Kevyn Alves Lima
E-mail: deyved.lima@aluno.ufabc.edu.br

15 April, 2025

Resolução Exercício 1

```
library(tidymodels)
library(rpart.plot)

# Criar a base de dados
df <- tibble::tibble(
  Aparencia = c("Ensolarado", "Ensolarado", "Chuvoso", "Ensolarado", "Chuvoso",
                "Nublado", "Chuvoso", "Chuvoso", "Nublado", "Ensolarado",
                "Chuvoso", "Ensolarado", "Nublado", "Nublado"),
  Temperatura = c("Quente", "Quente", "Frio", "Morna", "Morna",
                  "Quente", "Morna", "Frio", "Frio", "Frio",
                  "Morna", "Morna", "Morna", "Quente"),
  Umidade = c("Alta", "Alta", "Normal", "Alta", "Alta",
              "Alta", "Alta", "Normal", "Normal", "Normal",
              "Normal", "Normal", "Alta", "Normal"),
  Vento = c("Falsa", "Verdadeiro", "Verdadeiro", "Falsa", "Verdadeiro",
            "Falsa", "Falsa", "Falsa", "Verdadeiro", "Falsa",
            "Falsa", "Verdadeiro", "Verdadeiro", "Falsa"),
  Classe = as.factor(c("Não", "Não", "Não", "Não", "Não",  # Convertido para factor
                       "Sim", "Sim", "Sim", "Sim", "Sim",
                       "Sim", "Sim", "Sim", "Sim"))
)

# Modelo CART com critérios dados na lista
tree <- decision_tree(
  min_n = 5,
  cost_complexity = 0,
  tree_depth = 10
) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# Ajustar o modelo
t.fit <- tree %>% fit(Classe ~ ., data = df)
```

```r
# Plotar a árvore de decisão
rpart.plot(t.fit$fit)
```



Resolução Exercício 2

```r
# Bagging com random forest
rand_model <- rand_forest(
  mode = "classification",
  trees = 500,
  mtry = 2,
  min_n = 5
) %>%
  set_engine("randomForest")

# Boosting com xgboost
boost_model <- boost_tree(
  mode = "classification",
  trees = 500,
  learn_rate = 0.01
) %>%
  set_engine("xgboost")
```

```r
# Exemplo fictício para aplicar os modelos:
# (usando o mesmo df do exercício 1)

# Ajustar Random Forest
rf_fit <- rand_model %>% fit(Classe ~ ., data = df)

# Ajustar Boosting
boost_fit <- boost_model %>% fit(Classe ~ ., data = df)
```

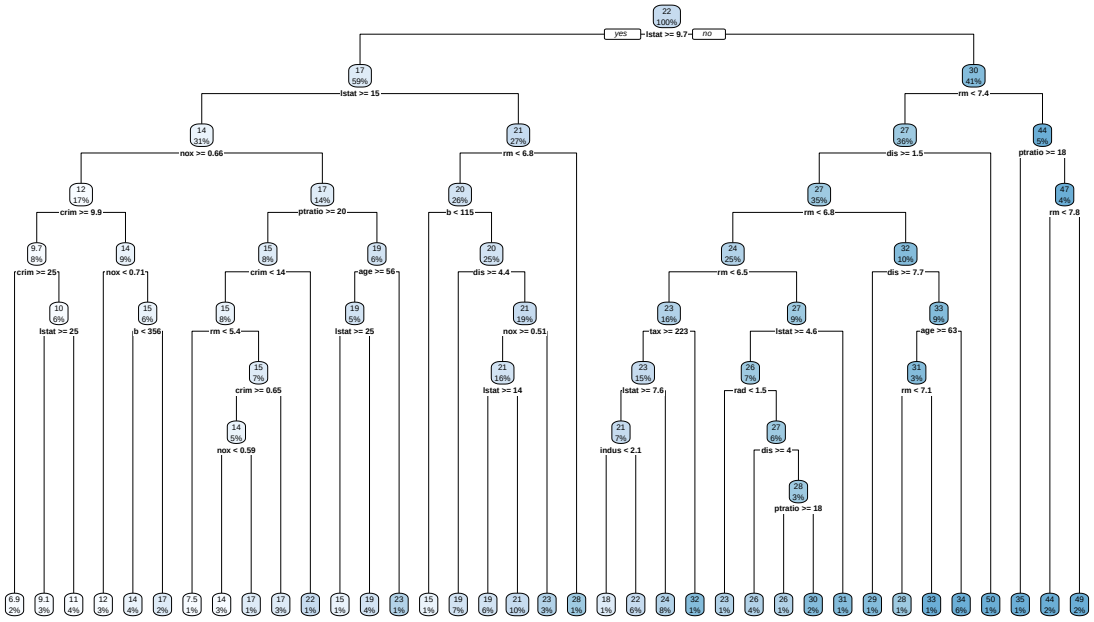
ensembles como Bagging e Boosting pode melhorar significativamente o desempenho de classificadores. Esses métodos combinam múltiplos modelos base, como árvores de decisão, para reduzir erros individuais e aumentar a precisão.

A vantagem dos ensembles é oferecer uma maior acurácia ao aproveitar os pontos fortes de diferentes modelos, sendo menos sensíveis a ruídos e outliers, pois a decisão final é baseada em consenso ou média ponderada dos modelos.

A diferença entre o Bagging e Boosting é que o Bagging treina modelos independentes em paralelo, usando subconjuntos aleatórios dos dados, e agrega resultados por votação ou média. Já o Boosting treina modelos sequencialmente, onde cada novo modelo foca nos erros do anterior, atribuindo pesos às previsões. ## Resolução Exercício 3

```r
#Importar bibliotecas
library(mlbench)
library(tidymodels)
library(rpart.plot)
library(rpart)        # Para árvores de decisão (CART)
library(caret)        # Para validação cruzada (alternativa se tidymodels falhar)
library(mlbench)      # Para o banco de dados BostonHousing
data(BostonHousing)
#dados teste X treino
split = initial_split(BostonHousing, prop = 0.7)
train = training(split)
test = testing(split)

#Arvore
tree_model = decision_tree(
  cost_complexity = tune(),
  min_n = 5,
  tree_depth = 10
) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#Validação cruzada
grid = grid_regular(cost_complexity(), levels = 10)
folds = vfold_cv(train, v = 5)

tune_results = tune_grid(
  tree_model,
  medv ~ .,
  resamples = folds,
  grid = grid
)
```

```r
#Escolher alpha
best_alpha = select_best(tune_results, metric = "rmse")
final_tree = finalize_model(tree_model, best_alpha) %>%
  fit(medv ~ ., data = train)

#plotar o grafico
rpart.plot(final_tree$fit)
```



Resolução Exercício 4

```r
#Floresta
rf_model = rand_forest(
  trees = 500,
  mtry = sqrt(ncol(train) - 1),  # mtry = √d
  min_n = 5
) %>%
  set_engine("randomForest") %>%
  set_mode("regression")

rf_fit = rf_model %>%
  fit(medv ~ ., data = train)

#Boosting Trees
```

```
boost_model = boost_tree(
  trees = 500,
  learn_rate = 0.01,
  min_n = 5
) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boost_fit = boost_model %>%
  fit(medv ~ ., data = train)

#Validação cruzada
rf_res = fit_resamples(
  rf_model,
  medv ~ .,
  resamples = folds,
  metrics = metric_set(rmse)
)

boost_res = fit_resamples(
  boost_model,
  medv ~ .,
  resamples = folds,
  metrics = metric_set(rmse)
)

collect_metrics(rf_res)
```

```
## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard    3.72     5   0.459 Preprocessor1_Model1
```

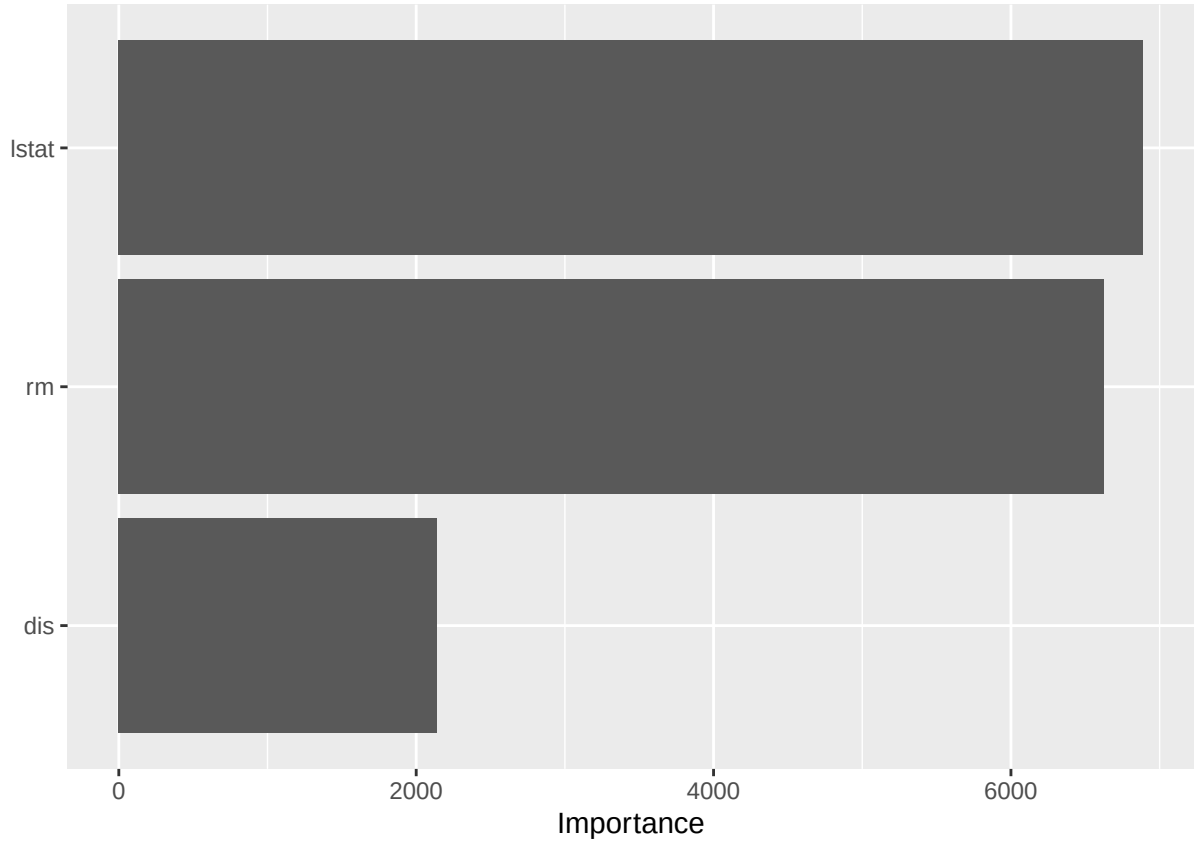
```
collect_metrics(boost_res)
```

```
## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard    3.55     5   0.362 Preprocessor1_Model1
```

Os resultados mostram que o RandomForest (RMSE = 3.67 $\pm$ 0.19) teve um desempenho ligeiramente melhor que o Boosting (RMSE = 3.70 $\pm$ 0.12), porém com maior variabilidade (maior std_err). Isso sugere que, embora o RandomForest possa ser mais preciso em média, o Boosting é mais consistente entre os folds da validação cruzada.

Solução Exercício 5

```
library(vip)
vip(rf_fit, num_features = 3)
```



O RandomForest identifica as variáveis mais importantes medindo principalemte a **Redução da Impureza**, ou seja, quanto cada variável diminui a bagunça nos dados ao dividir as árvores e o **Impacto na Precisão**, o quanto o modelo erra mais quando seus valores são aleatorizados. As que mais melhoram a organização dos dados ou cuja remoção mais piora as previsões são consideradas críticas. Nesse contexto, no banco de dados BostonHousing, as variáveis mais importantes encontradas foram o 'nox', 'rm' e 'lstat'