# Terceira Atividade

Rennan Guimarães

24/09/2024

## Contents

**Introdução** **2**

**Pré processamento e configurações iniciais** **2**

Dividindo os Dados em Treino e Teste . . . . . 2

Análise de Correlação e Seleção de Variáveis . . . . . 2

Definindo os Folds para Validação Cruzada . . . . . 2

Receitas de Pré-processamento . . . . . 2

Função para Coletar Métricas de Resample . . . . . 3

**Técnicas** **3**

Técnica 1: KNN . . . . . 3

Técnica 2: Árvore de Decisão . . . . . 4

Técnica 3: Random Forest . . . . . 5

Técnica 4: LightGBM . . . . . 5

Técnica 5: XGBoost . . . . . 6

**Avaliando resultados do treinamento** **7**

Média e Variância das Métricas . . . . . 7

Considerações . . . . . 9

Conclusão . . . . . 9

**Teste** **10**

Ajuste Final e Avaliação . . . . . 10

Compilando as Métricas do Conjunto de Teste . . . . . 11

Análise das Curvas ROC . . . . . 12

Conclusão . . . . . 12

**Selecionando o melhor modelo** **12**

# Introdução

O objetivo desse trabalho é comparar técnicas de classificação para identificar a melhor opção para a base de dados german.

Para isso, apurei 5 principais métricas: acurácia, f1-score (f_meass), precisão, recall e roc auc.

Para essas méticas, foram utilizadas 5 técnicas: KNN, Decision tree, LightGBM, Random Forest e XGBoost.

Caso queira partir diretamente para os resultados, intereja com o sumário acima.

Caso esteja queira acessar o código fonte ou a versão em HTML, acesse o repositório da atividade e os arquivos para essa atividade serão os nomeados como “final_clasificacao”.

# Pré processamento e configurações iniciais

## Dividindo os Dados em Treino e Teste

```
set.seed(17)
data_split <- initial_split(german_data, prop = 0.8, strata = "Good_loan")
train_data <- training(data_split)
test_data <- testing(data_split)
```

## Análise de Correlação e Seleção de Variáveis

```
numeric_vars <- train_data |>
  select(where(is.numeric))

correlation_matrix <- cor(numeric_vars, use = "complete.obs")

high_cor_vars <- findCorrelation(correlation_matrix, cutoff = 0.9, names = TRUE)
train_data <- train_data |> select(-all_of(high_cor_vars))
test_data <- test_data |> select(-all_of(high_cor_vars))
```

## Definindo os Folds para Validação Cruzada

```
set.seed(17)
folds <- vfold_cv(train_data, v = 10, strata = "Good_loan")
```

## Receitas de Pré-processamento

```
rec <- recipe(Good_loan ~ ., data = train_data) |>
  # Imputação
  step_impute_mode(all_nominal_predictors()) |>
  step_impute_median(all_numeric_predictors()) |>
  # Tratamento de outliers usando winsorization
  step_mutate_at(all_numeric_predictors(), fn = ~scales::squish(.x, quantile(.x, c(0.01, 0.99), na.rm =
  # Codificação
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |>
  # Remover variáveis com variância zero
  step_zv(all_predictors()) |>
  # Tratamento de desbalanceamento
  step_smote(Good_loan)
```

```
rec_normalized <- recipe(Good_loan ~ ., data = train_data) |>
  # Imputação
  step_impute_mode(all_nominal_predictors()) |>
  step_impute_median(all_numeric_predictors()) |>
  # Tratamento de outliers usando winsorization
  step_mutate_at(all_numeric_predictors(), fn = ~scales::squish(.x, quantile(.x, c(0.01, 0.99), na.rm =
  # Codificação
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |>
  # Remover variáveis com variância zero
  step_zv(all_predictors()) |>
  # Normalização
  step_normalize(all_numeric_predictors()) |>
  # Tratamento de desbalanceamento
  step_smote(Good_loan)
```

### Função para Coletar Métricas de Resample

```
collect_resample_metrics <- function(tune_results, best_params, model_name) {
  tune_results |>
    collect_metrics(summarize = FALSE) |>
    inner_join(best_params, by = names(best_params)) |>
    mutate(Model = model_name)
}
```

## Técnicas

### Técnica 1: KNN

```
knn_model <- nearest_neighbor(
  neighbors = tune(),
  weight_func = tune(),
  dist_power = tune()
) |>
  set_engine("kknn") |>
  set_mode("classification")
```

```
workflow_knn <- workflow() |>
  add_model(knn_model) |>
  add_recipe(rec)

knn_params <- parameters(knn_model)
```

```
## Warning: ‘parameters.model_spec()‘ was deprecated in tune 0.1.6.9003.
## i Please use ‘hardhat::extract_parameter_set_dials()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
set.seed(17)
knn_grid <- grid_random(knn_params, size = 50)

knn_results <- tune_grid(
  workflow_knn,
  resamples = folds,
  grid = knn_grid,
  metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
)

best_knn <- select_best(knn_results, metric = "roc_auc")

knn_resample_metrics <- collect_resample_metrics(knn_results, best_knn, "KNN")
```

### Técnica 2: Árvore de Decisão

```
dt_model <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()
) |>
  set_engine("rpart") |>
  set_mode("classification")

workflow_dt <- workflow() |>
  add_model(dt_model) |>
  add_recipe(rec)

dt_params <- parameters(dt_model)

set.seed(17)
dt_grid <- grid_random(dt_params, size = 50)

dt_results <- tune_grid(
  workflow_dt,
  resamples = folds,
  grid = dt_grid,
  metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
```

```
)

best_dt <- select_best(dt_results, metric = "roc_auc")

dt_resample_metrics <- collect_resample_metrics(dt_results, best_dt, "Árvore de Decisão")
```

### Técnica 3: Random Forest

```
rf_model <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) |>
  set_engine("ranger", importance = "impurity") |>
  set_mode("classification")

workflow_rf <- workflow() |>
  add_model(rf_model) |>
  add_recipe(rec)

rf_params <- parameters(rf_model) |>
  update(
    mtry = mtry(range = c(1, floor(sqrt(ncol(train_data))))),
    trees = trees(range = c(500, 2000)),
    min_n = min_n(range = c(2, 20))
  )

set.seed(17)
rf_grid <- grid_random(rf_params, size = 50)

rf_results <- tune_grid(
  workflow_rf,
  resamples = folds,
  grid = rf_grid,
  metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
)

best_rf <- select_best(rf_results, metric = "roc_auc")

rf_resample_metrics <- collect_resample_metrics(rf_results, best_rf, "Random Forest")
```

### Técnica 4: LightGBM

```
lightgbm_model <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  min_n = tune(),
```

```
  sample_size = tune(),
  mtry = tune()
) |>
  set_engine("lightgbm") |>
  set_mode("classification")

workflow_lightgbm <- workflow() |>
  add_model(lightgbm_model) |>
  add_recipe(rec)

lightgbm_params <- parameters(lightgbm_model) |>
  update(
    trees = trees(range = c(500, 2000)),
    tree_depth = tree_depth(range = c(1, 15)),
    learn_rate = learn_rate(range = c(0.01, 0.3)),
    loss_reduction = loss_reduction(),
    min_n = min_n(range = c(2, 20)),
    sample_size = sample_prop(range = c(0.5, 1)),
    mtry = mtry(range = c(1, floor(sqrt(ncol(train_data)))))
  )

set.seed(17)
lightgbm_grid <- grid_random(lightgbm_params, size = 50)

lightgbm_results <- tune_grid(
  workflow_lightgbm,
  resamples = folds,
  grid = lightgbm_grid,
  metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
)
```

```
## > A | warning: While computing binary `precision()`, no predicted events were detected (i.e.
##                `true_positive + false_positive = 0`).
##                Precision is undefined in this case, and `NA` will be returned.
##                Note that 24 true event(s) actually occurred for the problematic event level,
##                no

## There were issues with some computations   A: x1There were issues with some computations   A: x2There
```

```
best_lightgbm <- select_best(lightgbm_results, metric = "roc_auc")

lightgbm_resample_metrics <- collect_resample_metrics(lightgbm_results, best_lightgbm, "LightGBM")
```

### Técnica 5: XGBoost

```
xgb_model <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
```

```
  sample_size = tune(),
  mtry = tune(),
  min_n = tune()
) |>
  set_engine("xgboost") |>
  set_mode("classification")

workflow_xgb <- workflow() |>
  add_model(xgb_model) |>
  add_recipe(rec_normalized)

xgb_params <- parameters(xgb_model) |>
  update(
    trees = trees(range = c(500, 2000)),
    tree_depth = tree_depth(range = c(1, 15)),
    learn_rate = learn_rate(range = c(0.01, 0.3)),
    loss_reduction = loss_reduction(),
    sample_size = sample_prop(range = c(0.5, 1)),
    mtry = mtry(range = c(1, floor(sqrt(ncol(train_data))))),
    min_n = min_n(range = c(2, 20))
  )

set.seed(17)
xgb_grid <- grid_random(xgb_params, size = 50)

xgb_results <- tune_grid(
  workflow_xgb,
  resamples = folds,
  grid = xgb_grid,
  metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
)
```

```
## > A | warning: While computing binary `precision()`, no predicted events were detected (i.e.
##                `true_positive + false_positive = 0`).
##                Precision is undefined in this case, and `NA` will be returned.
##                Note that 24 true event(s) actually occurred for the problematic event level,
##                no

## There were issues with some computations   A: x1There were issues with some computations   A: x2There
```

```
best_xgb <- select_best(xgb_results, metric = "roc_auc")

xgb_resample_metrics <- collect_resample_metrics(xgb_results, best_xgb, "XGBoost")
```

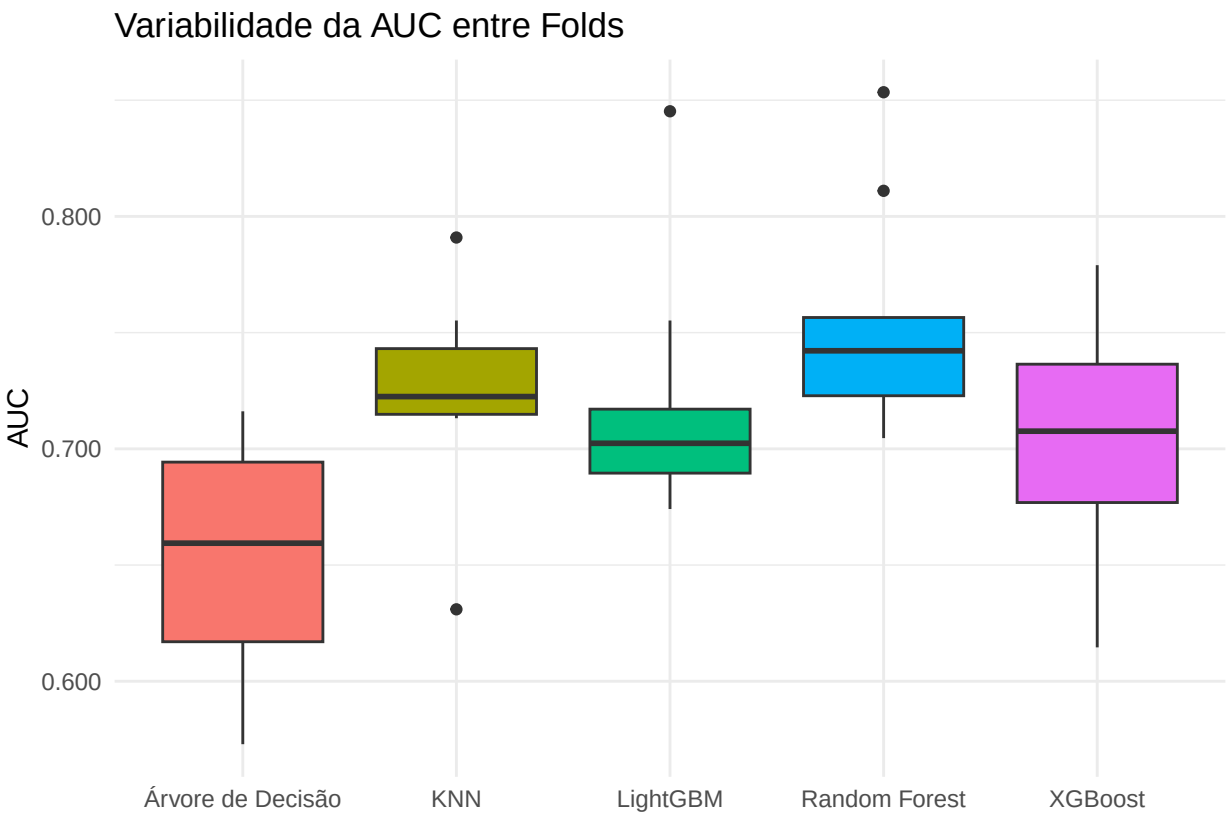
# Avaliando resultados do treinamento

## Média e Variância das Métricas

```
print(metrics_table)
```

```
## # A tibble: 5 x 6
##   Model              accuracy    f_meas      precision   recall      roc_auc
##   <chr>              <glue>      <glue>      <glue>      <glue>      <glue>
## 1 KNN                0.65 ± 0.04 0.53 ± 0.04 0.45 ± 0.04 0.67 ± 0.07 0.73 ± 0.04
## 2 LightGBM           0.72 ± 0.03 0.48 ± 0.07 0.53 ± 0.05 0.43 ± 0.08 0.72 ± 0.05
## 3 Random Forest      0.75 ± 0.03 0.42 ± 0.05 0.72 ± 0.17 0.31 ± 0.06 0.75 ± 0.05
## 4 XGBoost            0.71 ± 0.03 0.45 ± 0.06 0.52 ± 0.06 0.4 ± 0.08  0.71 ± 0.05
## 5 Árvore de Decisão 0.66 ± 0.05 0.43 ± 0.06 0.45 ± 0.08 0.43 ± 0.09 0.66 ± 0.05
```

```r
ggplot(resample_auc, aes(x = Model, y = .estimate, fill = Model)) +
  geom_boxplot(show.legend = FALSE) +
  theme_minimal() +
  labs(title = "Variabilidade da AUC entre Folds", y = "AUC", x = "") +
  scale_y_continuous(labels = scales::number_format(accuracy = 0.001))
```




1. **Acurácia**:
   - Random Forest mantém a melhor acurácia ($0.75 \pm 0.03$), seguido pelo LightGBM ($0.72 \pm 0.03$).
2. **F-measure**:
   - KNN apresenta o melhor F1-score ($0.53 \pm 0.04$), sugerindo um bom equilíbrio entre precisão e recall.
   - Entretanto, o boxplot mostra que KNN tem uma variabilidade considerável na AUC entre folds.
3. **Precisão**:

- Random Forest mantém a maior precisão ($0.72 \pm 0.17$), mas a alta variabilidade é um ponto de atenção.
- O boxplot corrobora esta variabilidade, mostrando uma ampla distribuição de AUC para Random Forest.

4. **Recall**:
   - KNN continua com o melhor recall ($0.67 \pm 0.07$).
   - O boxplot mostra que, apesar do bom recall, KNN tem uma distribuição de AUC mais baixa que outros modelos.
5. **ROC AUC**:
   - Random Forest e KNN lideram ($0.75 \pm 0.05$ e $0.73 \pm 0.04$, respectivamente).
   - O boxplot revela que Random Forest tem a distribuição de AUC mais alta, mas também a maior variabilidade.

## Considerações

1. **Trade-offs**:
   - O trade-off entre precisão e recall permanece evidente, especialmente para Random Forest e KNN.
   - O boxplot sugere que este trade-off também se reflete na estabilidade do desempenho entre folds.
2. **Consistência**:
   - LightGBM e XGBoost mostram desempenho consistente nas métricas tabulares.
   - O boxplot confirma esta consistência, mostrando distribuições de AUC mais compactas para estes modelos.
3. **Contexto de Negócio**:
   - A escolha entre priorizar recall (KNN) ou precisão (Random Forest) deve considerar não apenas as médias, mas também a variabilidade mostrada no boxplot.
4. **Variabilidade**:
   - A alta variabilidade do Random Forest, visível tanto nas métricas tabulares quanto no boxplot, reforça a necessidade de cautela ao considerá-lo como o melhor modelo.
   - O boxplot evidencia que modelos como LightGBM e XGBoost oferecem um equilíbrio melhor entre desempenho e estabilidade.

## Conclusão

Considerando tanto as métricas tabulares quanto a variabilidade da AUC entre folds:

1. LightGBM emerge como uma escolha mais robusta. Ele oferece um bom equilíbrio entre desempenho (segunda melhor acurácia) e consistência (distribuição de AUC compacta no boxplot).
2. XGBoost, embora com métricas ligeiramente inferiores ao LightGBM, também demonstra boa estabilidade e pode ser uma alternativa sólida.
3. Random Forest, apesar do melhor desempenho médio em algumas métricas, mostra alta variabilidade, o que pode ser um risco em aplicações práticas.
4. KNN, embora tenha bom recall, apresenta performance inferior em outras métricas e variabilidade considerável no boxplot.

Para o contexto da base German Credit, onde o equilíbrio e a consistência são cruciais, o **LightGBM se destaca como a opção mais promissora.** Ele oferece um bom compromisso entre identificar corretamente bons e maus riscos de crédito, mantendo um desempenho estável entre diferentes subconjuntos dos dados.

# Teste

## Ajuste Final e Avaliação

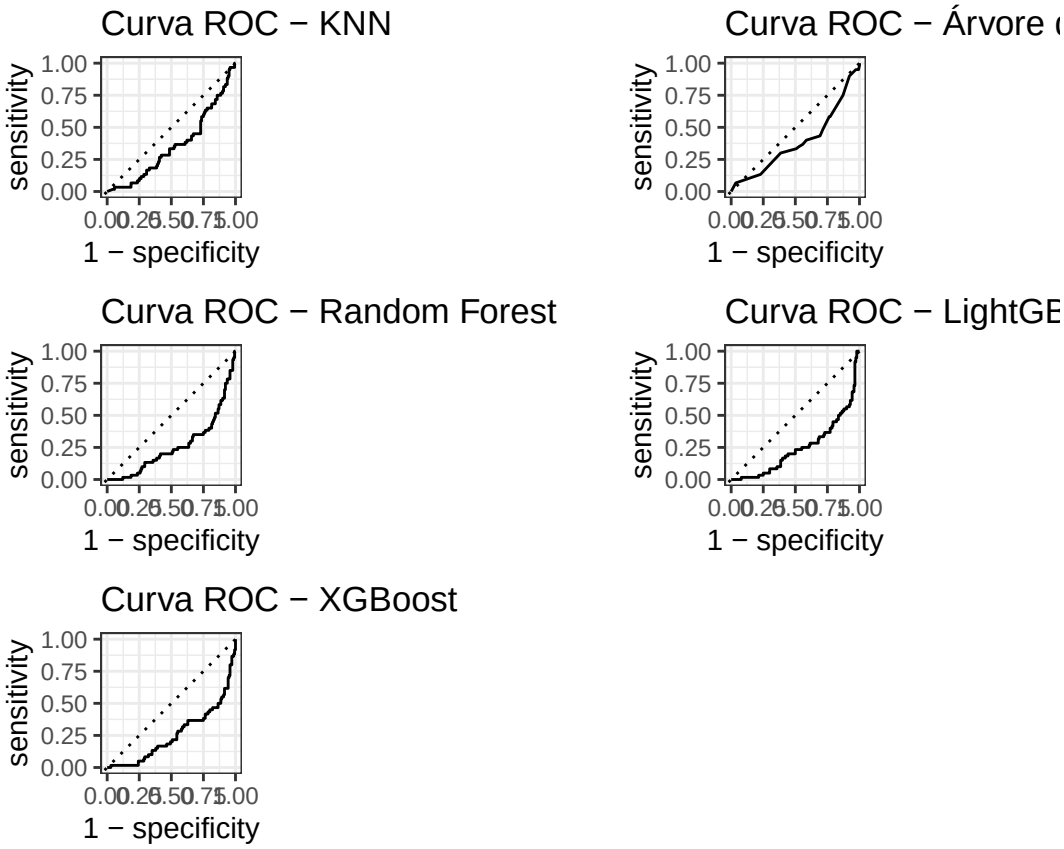
```
# Ajuste final para cada modelo
final_workflow_knn <- finalize_workflow(workflow_knn, best_knn)
final_workflow_rf <- finalize_workflow(workflow_rf, best_rf)
final_workflow_lightgbm <- finalize_workflow(workflow_lightgbm, best_lightgbm)
final_workflow_xgb <- finalize_workflow(workflow_xgb, best_xgb)
final_workflow_dt <- finalize_workflow(workflow_dt, best_dt)

# Lista de workflows finais
final_workflows <- list(
  "KNN" = final_workflow_knn,
  "Árvore de Decisão" = final_workflow_dt,
  "Random Forest" = final_workflow_rf,
  "LightGBM" = final_workflow_lightgbm,
  "XGBoost" = final_workflow_xgb
)

# Função para ajustar o modelo final e coletar métricas
extract_final_metrics <- function(final_workflow, model_name) {
  last_fit_result <- last_fit(
    final_workflow,
    split = data_split,
    metrics = metric_set(roc_auc, accuracy, yardstick::precision, yardstick::recall, f_meas)
  )

  metrics <- collect_metrics(last_fit_result) |>
    mutate(Model = model_name)

  predictions <- collect_predictions(last_fit_result)

  confusion <- predictions |>
    conf_mat(truth = Good_loan, estimate = .pred_class)

  roc_curve_data <- predictions |>
    roc_curve(truth = Good_loan, .pred_yes)

  roc_plot <- autoplot(roc_curve_data) +
    ggtitle(paste("Curva ROC -", model_name))

  list(metrics = metrics, confusion = confusion, roc_plot = roc_plot)
}

# Extrair métricas para cada modelo
results_list <- map2(final_workflows, names(final_workflows), extract_final_metrics)
```

### Compilando as Métricas do Conjunto de Teste

```
print(test_metrics)
```

```
## # A tibble: 5 x 6
##   Model              accuracy precision recall f_meas roc_auc
##   <chr>                 <dbl>     <dbl>  <dbl>  <dbl>   <dbl>
## 1 KNN                   0.62      0.409  0.6    0.486   0.642
## 2 Árvore de Decisão     0.655     0.437  0.517  0.473   0.612
## 3 Random Forest         0.735     0.621  0.3    0.404   0.737
## 4 LightGBM              0.745     0.588  0.5    0.541   0.750
## 5 XGBoost               0.73      0.55   0.55   0.55    0.743
```

```
do.call(grid.arrange, c(roc_plots, ncol = 2))
```




1. **Acurácia**:
   - LightGBM apresenta a melhor acurácia (0.745), seguido de perto por Random Forest (0.735) e XGBoost (0.730).
   - KNN tem a menor acurácia (0.620).
2. **Precisão**:
   - Random Forest lidera com 0.6207, seguido por LightGBM (0.5882).
   - KNN tem a menor precisão (0.4091).

3. **Recall**:
    - KNN tem o maior recall (0.6000), seguido por XGBoost (0.5500).
    - Random Forest tem o menor recall (0.3000).
4. **F-measure**:
    - XGBoost tem o melhor F1-score (0.5500), seguido de perto por LightGBM (0.5405).
    - Random Forest tem o menor F1-score (0.4045).
5. **ROC AUC**:
    - LightGBM tem o maior ROC AUC (0.7499), seguido por XGBoost (0.7431) e Random Forest (0.7368).
    - Árvore de Decisão tem o menor ROC AUC (0.6118).

### Análise das Curvas ROC

- **LightGBM e XGBoost**: Apresentam as curvas ROC mais próximas do canto superior esquerdo, indicando melhor desempenho geral.
- **Random Forest**: Mostra uma curva ROC forte, mas com algumas flutuações.
- **KNN e Árvore de Decisão**: Têm curvas ROC mais próximas da linha diagonal, indicando desempenho inferior em comparação com os outros modelos.

### Conclusão

1. **Melhor Desempenho Geral**: LightGBM e XGBoost se destacam como os modelos mais equilibrados e eficazes para este conjunto de dados. Eles apresentam boa acurácia, ROC AUC e um equilíbrio entre precisão e recall.
2. **Trade-offs**:
    - Random Forest mostra alta precisão, mas baixo recall, sugerindo que é conservador em suas previsões positivas.
    - KNN tem alto recall, mas baixa precisão, indicando uma tendência a prever mais positivos, mas com menor acurácia.
3. **Considerações Finais**: A escolha final do modelo para o desenvolvimento do restante do relatório será pelo modelo LightGBM, que obteve resultados consistentes entre as métricas e se destacou por uma acurácia e roc auc melhor.

## Selecionando o melhor modelo

```
# Selecionar o modelo com a maior AUC
best_model_name <- test_metrics |>
  filter(roc_auc == max(roc_auc)) |>
  pull(Model)

cat("O melhor modelo é:", best_model_name, "\n")
```

```
## O melhor modelo é: LightGBM
```

## Deploy

```
final_workflow_to_deploy <- final_workflows[[best_model_name]]

full_train_data <- bind_rows(train_data, test_data)

final_recipe <- recipe(Good_loan ~ ., data = full_train_data) |>
  # Imputação
  step_impute_mode(all_nominal_predictors()) |>
  step_impute_median(all_numeric_predictors()) |>
  # Tratamento de outliers usando winsorization
  step_mutate_at(all_numeric_predictors(), fn = ~scales::squish(.x, quantile(.x, c(0.01, 0.99), na.rm =
  # Codificação
  step_dummy(all_nominal_predictors(), one_hot = TRUE) |>
  # Remover variáveis com variância zero
  step_zv(all_predictors()) |>
  # Tratamento de desbalanceamento
  step_smote(Good_loan)

# Atualizar o workflow com a nova receita
final_workflow_to_deploy <- final_workflow_to_deploy |>
  update_recipe(final_recipe)

# Ajustar o modelo final
final_model <- final_workflow_to_deploy |> fit(data = full_train_data)
```

```
# Como o professor pediu para ser só um exemplo de deploy, vamos utilizar
# uma parte dos dados do conjunto que já utilizamos no treinamento, entretanto
# isso não deve ser feito na prática.
set.seed(17)
new_data <- full_train_data |>
  sample_n(10) |>
  select(-Good_loan)

predictions <- predict(final_model, new_data, type = "prob")
print(predictions)
```

```
## # A tibble: 10 x 2
##    .pred_no .pred_yes
##       <dbl>     <dbl>
##  1   0.360      0.640
##  2   0.428      0.572
##  3   0.126      0.874
##  4   0.391      0.609
##  5   0.0737     0.926
##  6   0.610      0.390
##  7   0.123      0.877
##  8   0.120      0.880
##  9   0.0624     0.938
## 10   0.761      0.239
```

```r
results <- bind_cols(
  new_data,
  predictions
)
print(results)
```

```
## # A tibble: 10 x 23
##    Status_of_existing_c~1 Duration_in_month Credit_history Purpose Credit_amount
##    <ord>                            <dbl> <chr>          <chr>           <dbl>
##  1 ... >= 200 DM                        9 no credits ta~ radio/~          1337
##  2 <NA>                                24 existing cred~ car (n~          7393
##  3 0 <= ... < 200 DM                   30 critical acco~ car (n~          2181
##  4 ... < 0 DM                          24 existing cred~ furnit~          2996
##  5 <NA>                                24 critical acco~ car (n~          1287
##  6 ... < 0 DM                          30 no credits ta~ busine~          8072
##  7 ... < 0 DM                           6 critical acco~ car (n~          4716
##  8 ... >= 200 DM                       24 existing cred~ furnit~          3749
##  9 ... >= 200 DM                       12 all credits a~ radio/~           409
## 10 0 <= ... < 200 DM                   30 existing cred~ furnit~          3441
## # i abbreviated name: 1: Status_of_existing_checking_account
## # i 18 more variables: Saving_account_bonds <ord>,
## #   Present_employment_since <ord>,
## #   Installment_rate_of_disposable_income <dbl>, Sex <chr>,
## #   Personal_status <chr>, Other_debtors_guarantors <chr>,
## #   Present_residence_since <dbl>, Property <chr>, Age_in_years <dbl>,
## #   Other_installment_plans <chr>, Housing <chr>, ...
```

```r
# Salvar o modelo para uso futuro
saveRDS(final_model, "final_classificacao_model.rds")
```