

FRAGRANCE

Modelo preditivo da avaliação de um perfume em profumo.com

Amanda Rodrigues Cunha*
Caio Cezar Veronezi Macedo†
Renato dos Santos Silva‡
Joao Victor Oliveira Correia de Brito§
Luis Guilherme Redigolo Crosselli¶

15 December, 2024

1 Introdução

Parfumo é um *website* que reúne conhecedores e entusiastas de fragâncias, organizando uma base de dados com pouco mais de 190 mil perfumes, criados por cerca de 12 mil marcas. Segundo palavras registradas na sua página inicial:

Parfumo is the home for all fragrance connoisseurs & enthusiasts!
Discover new perfumes, organize your collection, connect with other fragrance lovers and much more!

Ou seja, é também uma rede social em que os participantes compartilham opiniões em *forums* e registram suas avaliações sobre perfumes.

Neste projeto, exploramos o conjunto de dados The Scent of Data (**parfumo**), geradas a partir da base de dados do profumo.com e publicadas no TidyTuesday (quinquagésima semana). Na sua página de introdução, encontramos algumas perguntas de partida:

Quais fatores mais influenciam a avaliação de um perfume?
Há famílias de aromas específicas que dominam o mercado? Como são percebidas pelos usuários?
A popularidade de certas notas de fragâncias mudou ao longo do tempo?

A primeira exige uma abordagem de inferência, porquanto deseja-se encontrar a influência de certas características sobre a avaliação de um perfume. A terceira pede por uma análise exploratória. A segunda, porém, é passível de modelagem preditiva. Dadas certas características de um perfume, incluindo famílias olfativas, estamos interessados em prever sua avaliação. Tal modelo poderia ser usado, por exemplo, como instrumento para a projeção de aceitação de certo perfume em elaboração.

2 Análise e preparação dos dados

2.1 Descrição dos dados

A tabela abaixo lista as variáveis do conjunto de dados **parfumo**, junto de suas classes (R) e descrições:

*amanda.cunha@aluno.ufabc.edu.br

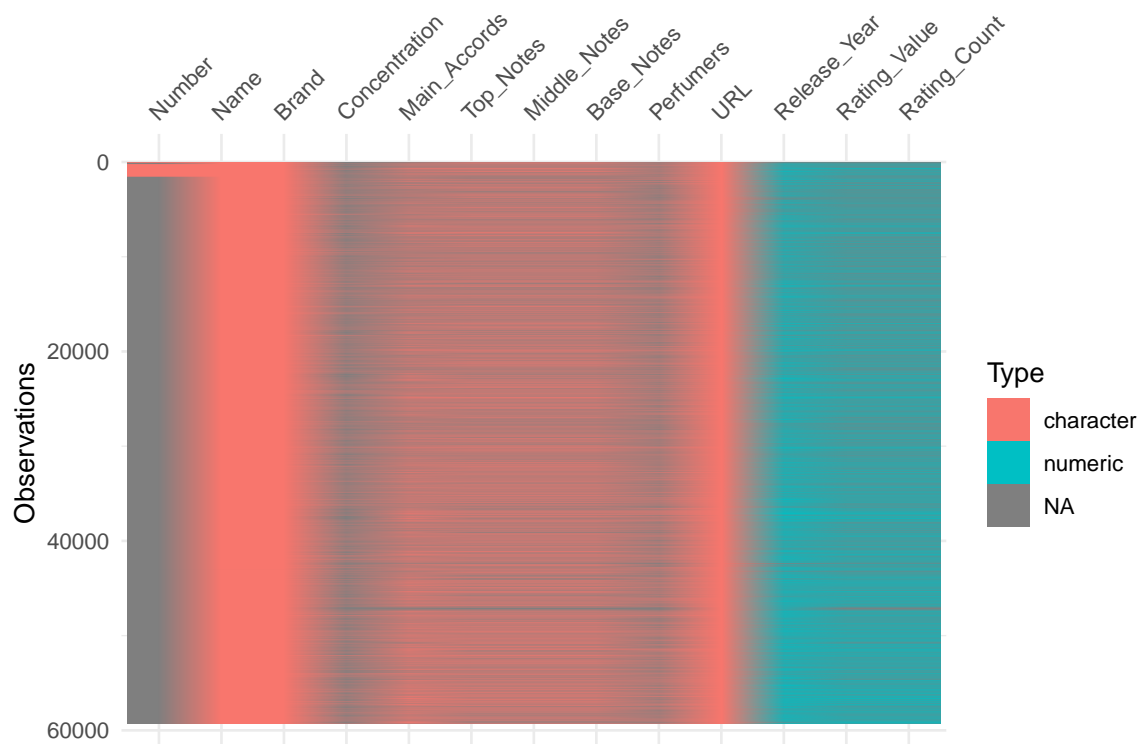
†cezar.veronezi@aluno.ufabc.edu.br

‡renato.santos@aluno.ufabc.edu.br

§brito.joao@aluno.ufabc.edu.br

¶luis.crosselli@aluno.ufabc.edu.br

Variável	Classe	Tipo	Descrição
Number	character	nominal	Identificador único atribuído a cada perfume.
Name	character	nominal	Nome do perfume ou fragrância.
Brand	character	nominal	Marca ou fabricante da fragrância.
Release_Year	double	discreta	Ano em que a fragrância foi lançada.
Concentration	character	ordinal	Concentração da fragrância.
Rating_Value	double	discreta	Pontuação geral atribuída pelos usuários.
Rating_Count	double	discreta	Número de avaliações de usuários para a fragrância.
Main_Accords	character	nominal	Principais características ou acordes olfativos da fragrância.
Top_Notes	character	nominal	Notas iniciais percebidas logo após a aplicação.
Middle_Notes	character	nominal	Notas médias ou de coração da fragrância, que surgem após as notas de topo.
Base_Notes	character	nominal	Notas finais e duradouras que permanecem após a fragrância secar.
Perfumers	character	nominal	Criadores ou perfumistas responsáveis pela composição da fragrância.
URL	character	nominal	Link para a página do produto no Parfumo.com.



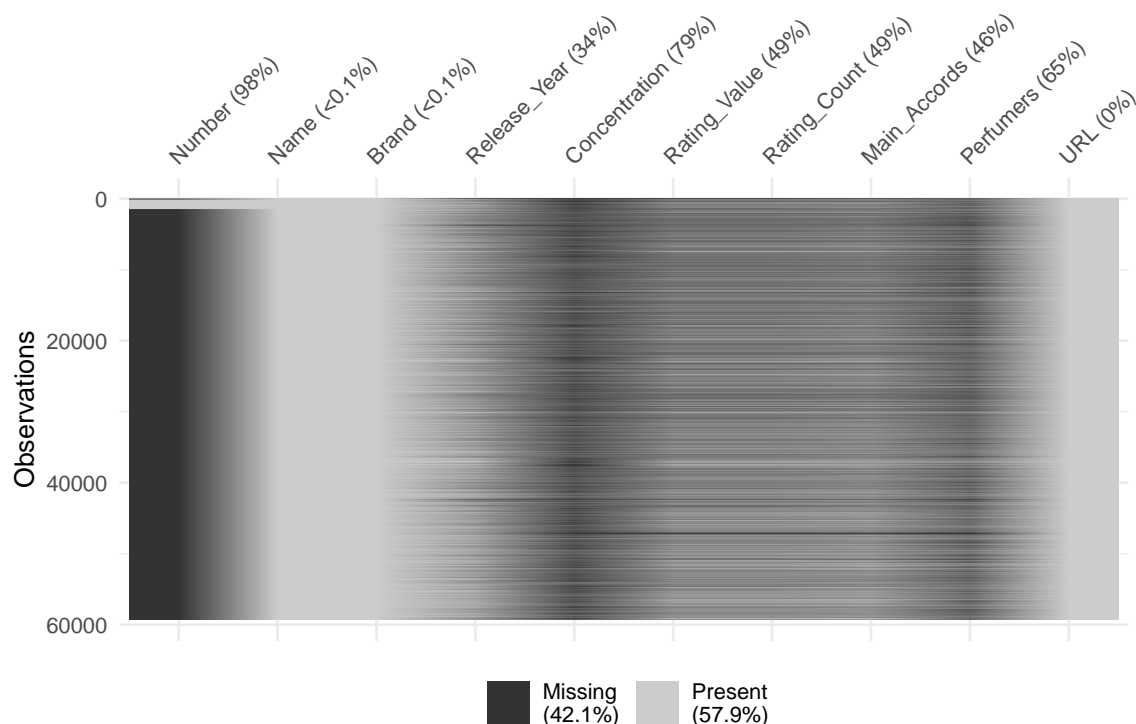
2.2 Análise de *missings* e *data cleaning*

Primeiramente, vamos analisar os possíveis valores para as variáveis olfativas, ou seja, quantas notas distintas podem ocorrer em cada uma:

Type	#Notes
Main_Accords	22
Top_Notes	2430
Middle_Notes	2629
Base_Notes	1835

Com exceção de **Main_Accords**, com 22 notas de fragância, as variáveis olfativas possuem um domínio de notas na casa de 10^3 possibilidades. Com isso, vamos restringir nosso modelo às características principais de cada perfume.

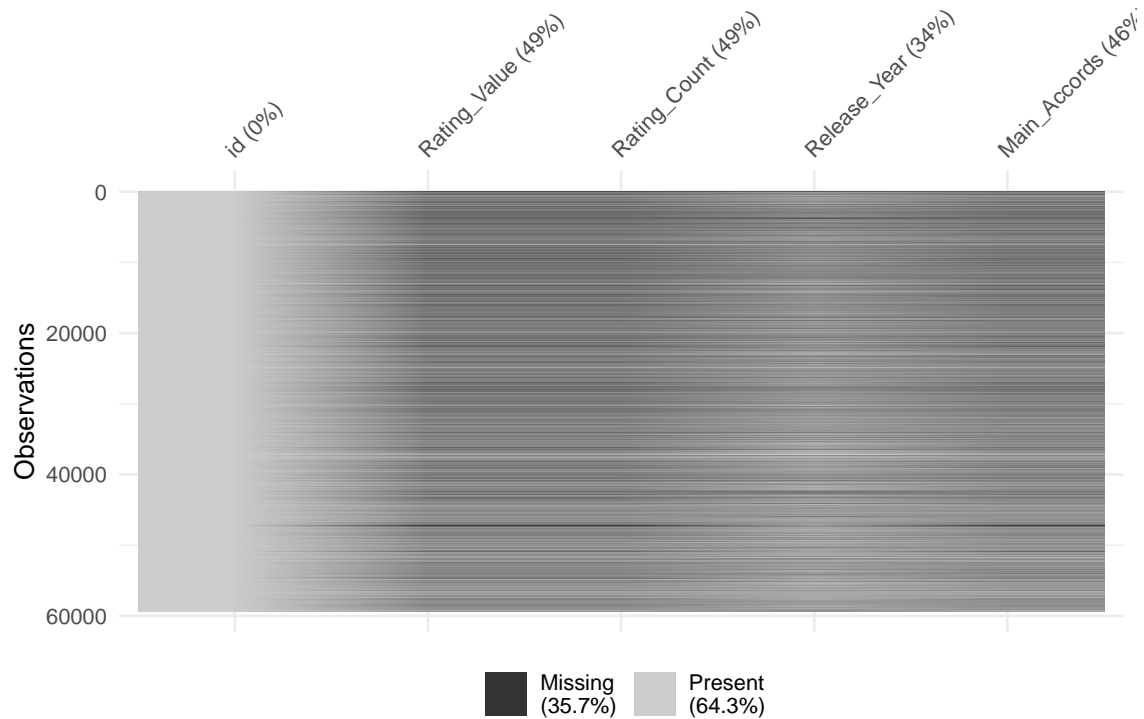
O gráficos abaixo resumem a ocorrência de *missings* no conjunto de dados:



Note que **Main_Accords** ocorre apenas em 54% dos dados. Além disso, **Concentration** está ausente em 79% deles, e **Perfumers** em 65%. Apenas **Release_Year**, **Rating_Count**, **Rating_Value** e **Brand** possuem presença que talvez não seja limitante à **Main_Notes** (Note que **Number** é irrelevante).

Vamos usar **Name**, **Brand** e **URL** como chave primária, e filtrar **Concentration** e **Perfumers** dos dados. (Lembrando que **Brand** possui domínio da ordem de 10^3 possibilidades, vamos ignorá-la porquanto pode fragmentar excessivamente os dados):

Agora, temos o seguinte gráfico de *missings*:



Vamos alterar `Release_Year` para `Years_Older` por ser mais intuitivo:

```
df_years <- df_cleaned %>%
  mutate(Years_Older = as.integer(format(Sys.Date(), "%Y")) - Release_Year) %>%
  select(-Release_Year)
```

Ao filtrarmos os *missings* e transformarmos `Main_Accords` desdobrando-a em variáveis binárias, uma para cada nota olfativa via *one-hot encoding*, temos o seguinte resultado resumido:

Table 3: Data summary

Name	input_table
Number of rows	23389
Number of columns	26
Column type frequency:	
numeric	26
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id	0	1	31911.82	16812.12	1.0	17941.0	33030.0	46659.0	59286	
Note_Floral	0	1	0.58	0.49	0.0	0.0	1.0	1.0	1	
Note_Fresh	0	1	0.49	0.50	0.0	0.0	0.0	1.0	1	
Note_Fruity	0	1	0.38	0.48	0.0	0.0	0.0	1.0	1	
Note_Green	0	1	0.26	0.44	0.0	0.0	0.0	1.0	1	
Note_Spicy	0	1	0.54	0.50	0.0	0.0	1.0	1.0	1	
Note_Citrus	0	1	0.26	0.44	0.0	0.0	0.0	1.0	1	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Note_Earthy	0	1	0.06	0.23	0.0	0.0	0.0	0.0	1	
Note_Smoky	0	1	0.09	0.28	0.0	0.0	0.0	0.0	1	
Note_Woody	0	1	0.49	0.50	0.0	0.0	0.0	1.0	1	
Note_Aquatic	0	1	0.09	0.28	0.0	0.0	0.0	0.0	1	
Note_Creamy	0	1	0.13	0.33	0.0	0.0	0.0	0.0	1	
Note_Sweet	0	1	0.53	0.50	0.0	0.0	1.0	1.0	1	
Note_Synthetic	0	1	0.20	0.40	0.0	0.0	0.0	0.0	1	
Note_Animal	0	1	0.05	0.22	0.0	0.0	0.0	0.0	1	
Note_Oriental	0	1	0.17	0.37	0.0	0.0	0.0	0.0	1	
Note_Resinous	0	1	0.11	0.31	0.0	0.0	0.0	0.0	1	
Note_Leathery	0	1	0.07	0.26	0.0	0.0	0.0	0.0	1	
Note_Powdery	0	1	0.23	0.42	0.0	0.0	0.0	0.0	1	
Note_Gourmand	0	1	0.12	0.32	0.0	0.0	0.0	0.0	1	
Note_Chypre	0	1	0.04	0.19	0.0	0.0	0.0	0.0	1	
Note_Fougère	0	1	0.02	0.12	0.0	0.0	0.0	0.0	1	
Rating_Value	0	1	7.32	0.85	0.9	6.9	7.4	7.9	10	
Rating_Count	0	1	73.36	129.51	2.0	9.0	27.0	79.0	2732	
Years_Older	0	1	14.45	16.41	0.0	6.0	10.0	17.0	315	
Note_Count	0	1	4.87	0.52	1.0	5.0	5.0	5.0	5	

Agora, vamos realizar uma regressão linear para verificar a importância das variáveis:

```
data <- input_table %>%
  select(-id)

linear_model <- lm(Rating_Value ~ ., data = data)
print(tidy(linear_model), n = Inf)
```

```
## # A tibble: 25 x 5
##   term                estimate std.error statistic  p.value
##   <chr>              <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)        7.12      0.0476     150.      0
## 2 Note_Floral         0.0591    0.0871      0.678 4.98e- 1
## 3 Note_Fresh        -0.0772    0.0870     -0.887 3.75e- 1
## 4 Note_Fruity        -0.0522    0.0870     -0.600 5.49e- 1
## 5 Note_Green         0.134     0.0870      1.54 1.24e- 1
## 6 Note_Spicy         0.123     0.0870      1.41 1.57e- 1
## 7 Note_Citrus        0.0644    0.0870      0.740 4.59e- 1
## 8 Note_Earthy        0.0785    0.0892      0.880 3.79e- 1
## 9 Note_Smoky         0.120     0.0882      1.36 1.73e- 1
## 10 Note_Woody         0.100     0.0869      1.15 2.48e- 1
## 11 Note_Aquatic      -0.101     0.0881     -1.14 2.53e- 1
## 12 Note_Creamy       0.391     0.0874      4.47 7.76e- 6
## 13 Note_Sweet        0.0450    0.0869      0.518 6.05e- 1
## 14 Note_Synthetic   -0.520     0.0870     -5.97 2.35e- 9
## 15 Note_Animal       0.224     0.0893      2.51 1.21e- 2
## 16 Note_Oriental     0.256     0.0873      2.93 3.36e- 3
## 17 Note_Resinous     0.184     0.0877      2.10 3.56e- 2
## 18 Note_Leathery     0.223     0.0886      2.51 1.20e- 2
## 19 Note_Powdery      0.0934    0.0872      1.07 2.84e- 1
## 20 Note_Gourmand     0.0702    0.0880      0.797 4.25e- 1
## 21 Note_Chypre       0.421     0.0905      4.65 3.31e- 6
```

```
## 22 Note_Fougère      0.170      0.0943      1.81  7.07e- 2
## 23 Rating_Count      0.000575 0.0000391    14.7  1.01e-48
## 24 Years_Older       0.000982 0.000325     3.02  2.53e- 3
## 25 Note_Count       -0.0239   0.0868     -0.276 7.83e- 1
```

3 Modelagem

Nosso objetivo é encontrar $\hat{y} := f(X) + \epsilon$, em que \hat{y} é a predição para $y \in [0, 10] \subseteq \mathbb{R}$, a avaliação de um perfume. Logo, temos um problema de **regressão**. Para isso, vamos usar **Elastic Net**, **Gradient Boosting** e **Support Vector Machines**.

Para reproducibilidade, vamos fixar a semente usada para geração de números pseudoaleatórios:

```
seed <- 42
set.seed(seed)
```

Os dados são divididos em conjuntos de treinamento e teste com:

```
data <- input_table %>%
  select(-id)

data_split <- initial_split(data, prop = 0.8)
train_data <- training(data_split)
test_data <- testing(data_split)
```

Os dados precisam ser normalizados. Neste caso, apenas as variáveis numéricas não binárias:

```
recipe_spec <- recipe(Rating_Value ~ ., data = data) %>%
  # step_normalize(Rating_Count, Years_Older, Note_Count)
  step_normalize(all_predictors())
```

Para comparação, usamos os modelos **Naive Mean** e **Naive Median**, dados a seguir:

```
# Calculate naive mean
naive_mean <- train_data %>%
  summarise(naive_mean = mean(Rating_Value))

print(naive_mean)

## # A tibble: 1 x 1
##   naive_mean
##       <dbl>
## 1       7.32

# Calculate naive median
naive_median <- train_data %>%
  summarise(naive_median = median(Rating_Value))

print(naive_median)
```

```
## # A tibble: 1 x 1
##   naive_median
##       <dbl>
## 1       7.4

rmse_mean <- test_data %>%
  summarise(rmse_mean = sqrt(mean((Rating_Value - naive_mean$naive_mean)^2)))
```

```

# Calculate RMSE for naive median model
rmse_median <- test_data %>%
  summarise(rmse_median = sqrt(mean((Rating_Value - naive_median$naive_median)^2)))

print(rmse_mean)

## # A tibble: 1 x 1
##   rmse_mean
##   <dbl>
## 1      0.845

print(rmse_median)

## # A tibble: 1 x 1
##   rmse_median
##   <dbl>
## 1      0.849

```

3.1 Elastic Net

```

elastic_net_spec <-
  linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("regression")

workflow_spec <- workflow() %>%
  add_recipe(recipe_spec) %>%
  add_model(elastic_net_spec)

# Define a grid of hyperparameters to tune
grid_spec <- grid_regular(
  penalty(range = c(-5, 5)),      # Range for lambda (penalty)
  mixture(range = c(0, 1)),      # Range for alpha (mixture)
  levels = 10                    # 5 levels for each parameter
)

# Perform cross-validation for hyperparameter tuning
cv_folds <- vfold_cv(train_data, v = 5) # 5-fold cross-validation

# Tune the model using the grid of hyperparameters
tuned_results <- tune_grid(
  workflow_spec,
  resamples = cv_folds,
  grid = grid_spec,
  metrics = metric_set(rmse)
)

# View the best hyperparameters
best_params <- tuned_results %>%
  select_best()

print(best_params)

## # A tibble: 1 x 3
##   penalty mixture .config

```

```
##      <dbl>      <dbl> <chr>
## 1 0.00001    0.111 Preprocessor1_Model011

# Finalize the workflow with the best parameters
final_workflow <- workflow_spec %>%
  finalize_workflow(best_params)

# Fit the final model on the full training data
final_model <- fit(final_workflow, data = train_data)

# Evaluate the model on the test set
test_results <- predict(final_model, test_data) %>%
  bind_cols(test_data %>% select(Rating_Value)) %>%
  metrics(truth = Rating_Value, estimate = .pred)

print(test_results)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <dbl>
## 1 rmse    standard      0.741
## 2 rsq     standard      0.230
## 3 mae     standard      0.543
```

3.2 Gradient Boosting

```
# Define the Gradient Boosting model specification
boosting_spec <- boost_tree(
  trees = tune(),           # Number of trees
  learn_rate = tune(),      # Learning rate
  tree_depth = tune(),      # Maximum tree depth
  min_n = tune(),           # Minimum number of observations in a node
  loss_reduction = tune()   # Minimum loss reduction (gamma in xgboost)
) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

# Define the workflow
workflow_spec <- workflow() %>%
  add_recipe(recipe_spec) %>%
  add_model(boosting_spec)

# Define a grid of hyperparameters to tune
grid_spec <- grid_random(
  trees(range = c(50, 500)),      # Number of trees
  learn_rate(range = c(0.01, 0.3)), # Learning rate
  tree_depth(range = c(3, 10)),    # Maximum tree depth
  min_n(range = c(2, 20)),         # Minimum number of observations in a node
  loss_reduction(range = c(0, 10)), # Minimum loss reduction
  size = 10                       # Number of random combinations
)

# Create cross-validation folds
cv_folds <- vfold_cv(train_data, v = 5) # 5-fold cross-validation
```



```

# Tune the model using the hyperparameter grid
tuned_results <- tune_grid(
  workflow_spec,
  resamples = cv_folds,
  grid = grid_spec,
  metrics = metric_set(rmse) # Use RMSE as the evaluation metric
)

# View the tuning results
tuned_results

## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits          id   .metrics      .notes
##   <list>         <chr> <list>      <list>
## 1 <split [14968/3743]> Fold1 <tibble [10 x 9]> <tibble [0 x 3]>
## 2 <split [14969/3742]> Fold2 <tibble [10 x 9]> <tibble [0 x 3]>
## 3 <split [14969/3742]> Fold3 <tibble [10 x 9]> <tibble [0 x 3]>
## 4 <split [14969/3742]> Fold4 <tibble [10 x 9]> <tibble [0 x 3]>
## 5 <split [14969/3742]> Fold5 <tibble [10 x 9]> <tibble [0 x 3]>

# View the best hyperparameters
best_params <- tuned_results %>%
  select_best()

print(best_params)

## # A tibble: 1 x 6
##   trees min_n tree_depth learn_rate loss_reduction .config
##   <int> <int>    <int>      <dbl>      <dbl> <chr>
## 1   132   19        9        1.41        930. Preprocessor1_Model106

# Finalize the workflow with the best parameters
final_workflow <- workflow_spec %>%
  finalize_workflow(best_params)

# Fit the final model on the full training data
final_model <- fit(final_workflow, data = train_data)

# Evaluate the model on the test set
test_results <- predict(final_model, test_data) %>%
  bind_cols(test_data %>% select(Rating_Value)) %>%
  metrics(truth = Rating_Value, estimate = .pred)

print(test_results)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <dbl>
## 1 rmse   standard      0.790
## 2 rsq    standard      0.142
## 3 mae    standard      0.588

```

3.3 Support Vector Machines

```
# Define the SVM model specification
svm_spec <- svm_rbf(
  cost = tune(), # Cost (regularization parameter)
  rbf_sigma = tune() # Sigma (kernel parameter for RBF)
) %>%
  set_engine("kernlab") %>%
  set_mode("regression")

# Define the workflow
workflow_spec <- workflow() %>%
  add_recipe(recipe_spec) %>%
  add_model(svm_spec)

# Define a grid of hyperparameters to tune
grid_spec <- grid_regular(
  cost(range = c(-2, 2)), # Cost range (log scale)
  rbf_sigma(range = c(-2, 2)), # Sigma range (log scale)
  levels = 5 # Number of levels for each parameter
)

# Create cross-validation folds
cv_folds <- vfold_cv(train_data, v = 5) # 5-fold cross-validation

# Tune the model using the hyperparameter grid
tuned_results <- tune_grid(
  workflow_spec,
  resamples = cv_folds,
  grid = grid_spec,
  metrics = metric_set(rmse) # Use RMSE as the evaluation metric
)

# View the tuning results
tuned_results

## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits          id   .metrics      .notes
##   <list>         <chr> <list>      <list>
## 1 <split [14968/3743]> Fold1 <tibble [25 x 6]> <tibble [0 x 3]>
## 2 <split [14969/3742]> Fold2 <tibble [25 x 6]> <tibble [0 x 3]>
## 3 <split [14969/3742]> Fold3 <tibble [25 x 6]> <tibble [0 x 3]>
## 4 <split [14969/3742]> Fold4 <tibble [25 x 6]> <tibble [0 x 3]>
## 5 <split [14969/3742]> Fold5 <tibble [25 x 6]> <tibble [0 x 3]>

# View the best hyperparameters
best_params <- tuned_results %>%
  select_best()

print(best_params)

## # A tibble: 1 x 3
##   cost rbf_sigma .config
##   <dbl>      <dbl> <chr>
```

```
## 1      4      0.01 Preprocessor1_Model05
# Finalize the workflow with the best parameters
final_workflow <- workflow_spec %>%
  finalize_workflow(best_params)

# Fit the final model on the full training data
final_model <- fit(final_workflow, data = train_data)

# Evaluate the model on the test set
test_results <- predict(final_model, test_data) %>%
  bind_cols(test_data %>% select(Rating_Value)) %>%
  metrics(truth = Rating_Value, estimate = .pred)

print(test_results)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <dbl>
## 1 rmse    standard      0.719
## 2 rsq     standard      0.279
## 3 mae     standard      0.520
```

4 Análise dos resultados

5 Conclusões

6 Anexos