Projeto III

Iara Souza

19/04/2021

Introdução

Os genes são as unidades funcionais básicas do material genético dos seres vivos. Alguns desempenham funções tão importantes na célula e, em maior escala, no organismo, que mutações (alterações herdáveis) nocivas nestes podem levar à morte. Assim, podemos classificar os genes em essenciais (ou letais), os quais são genes cuja função deve ser preservada para a viabilidade do organismo e genes não-esssenciais (ou não-letais), os quais não causam a morte do organismo caso sejam alterados.

Os genes essenciais compartilham algumas características moleculares. Já se sabe que algumas propriedades de redes biológicas estão relacionadas à essencialidade de genes. É sabido que genes "hubs", genes que se comunicam com vários outros genes e que coordenam o fluxo de informação da rede, tenham uma maior probabilidade de serem essenciais. Propriedades como o grau e centralidade dos nós da rede podem indicar quais nós são mais importantes para manter a estrutura da rede biológica. Por isso, neste dataset, estas duas medidas foram tomadas para cada gene anotado em camudongo e levedura. Além disso, os genes essenciais são, em geral, mais antigos que os genes não-essenciais, implicando que aqueles surgiram em um ponto mais inicial da escala evolutiva e foram conservados ao longo da evolução em diferentes organismos. Por fim, os genes essenciais estão relacionados às funções moleculares básicas, as quais são indispensáveis para o bom funcionamento da célula e do organismo.

Objetivo

Aplicar os algoritmos de classificação Random Forest, SVM polinomial e SVM radial para a criação de um modelo de classificação de genes em essenciais e não esssenciais, usando dados de dois organismos modelo (*Mus musculus* - camundongo - e *Saccharomyces cerevisiae* - levedura).

Coleta de dados

Os dados usados para esta tarefa vêm de diversas fontes. O processo de obtenção pode ser resumido nos seguintes passos:

1. Anotação funcional dos genes. Realizou-se uma busca nos bancos de dados especializados dos organismos modelo, a fim de obter-se quais genes são essenciais e quais são não essenciais.

2. Inferência das raízes evolutivas. Para cada organismo, realizou-se uma análise evolutiva com a finalidade de se obter uma escala de ancestralidade (genes que surgiram mais recentemente ou mais antigamente).

3. Construção das redes de interação proteína-proteína para cada organismo. As redes biológicas foram construídas para possibilitar o estudo das propriedades dos genes essenciais e não-essenciais no contexto de um sistema.

As variáveis do conjunto de dados são:

- `ensembl_peptide_id`: identificador da proteína (gene);
- `cog_id`: identificador do grupo de ortólogos ao qual o gene pertence;
- `Root`: raiz evolutiva inferida. Quanto maior, mais antigo;
- `Dscore`: escore de consistência para a inferência da raiz;
- `Pvalue` e `AdjPvalue`: p-valor e p-valor ajustado para a inferência da raiz evolutiva;
- `abundance`: medida de abundância dos grupos de ortólogos;
- `diversity`: medida de diversidade dos grupos de ortólogos;
- `plasticity`: índice de plasticidade evolutiva;
- `ancestry`: ancestralidade do gene. Transformação da variável `Root`(escala de 0 - mais novo - a 1 - mais antigo);
- `btw`: centralidade do nó na rede;
- `degree`: grau do nó na rede.

Todos os dados estão disponíveis neste link https://github.com/iaradsouza1/essential_genes_paper.

```
load("eg.rda")
```

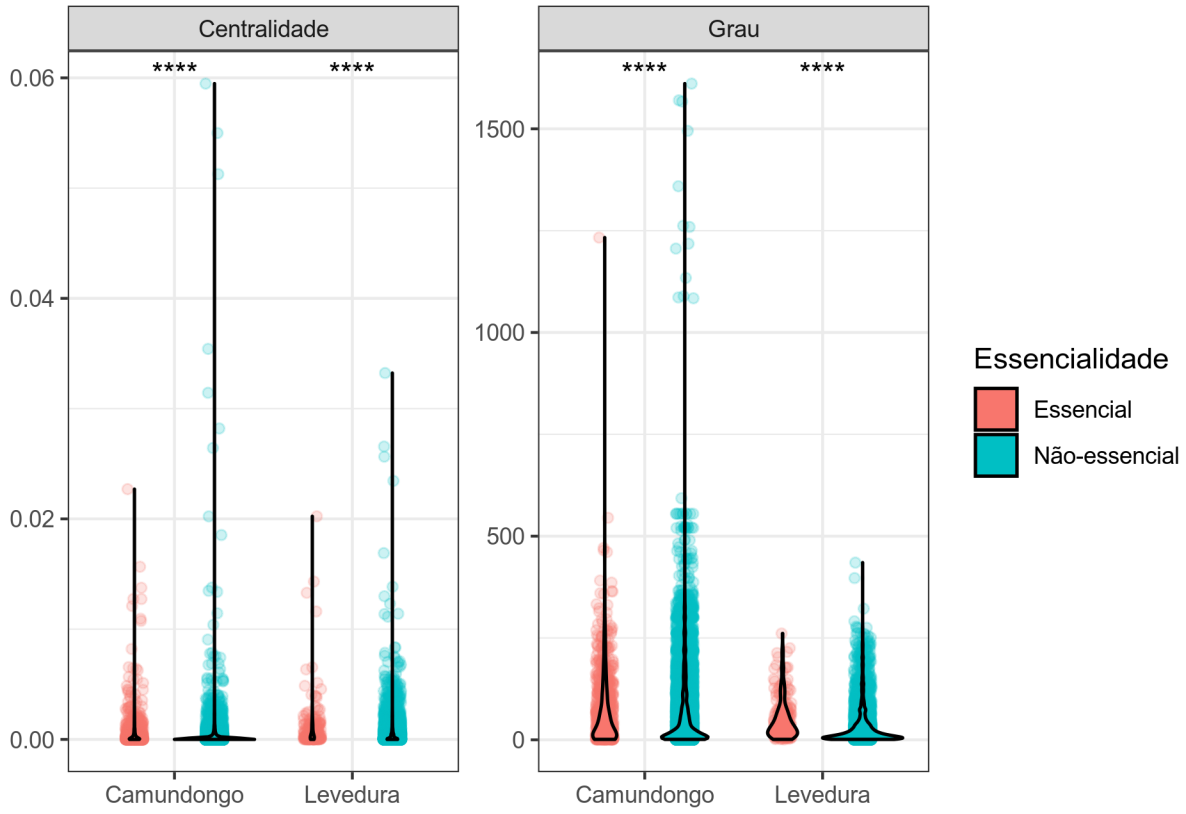
Resultados

Análise exploratória

Aqui iremos verificar algumas características iniciais dos perfis de essencialidade de camundongo e de levedura.

As propriedades de rede apresentam distribuições distintas entre as duas classes de essencialidade, em ambos os organismos.

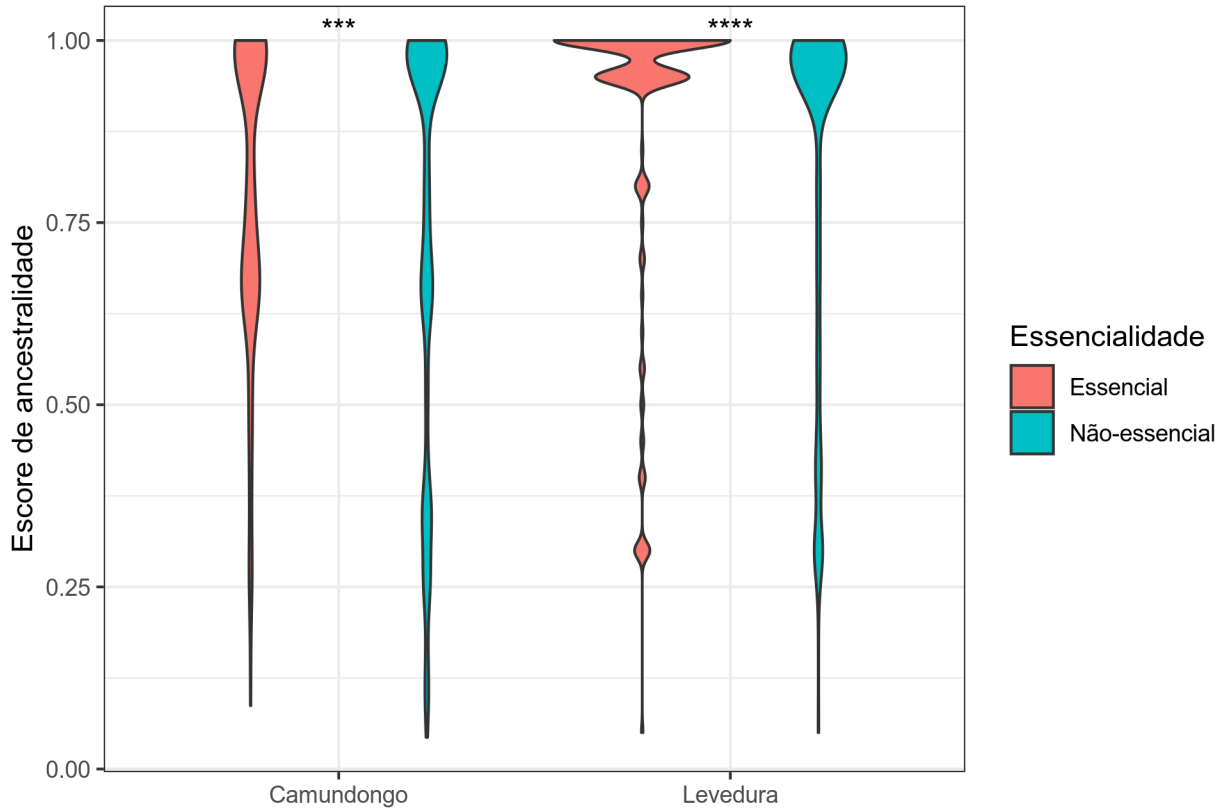
```
eg %>%
  gather(netprop, value, btw:degree) %>%
  ggplot(aes(x = organism, y = value,
             fill = lethal_nonlethal, col = lethal_nonlethal)) +
  geom_point(position = position_jitterdodge(jitter.width = 0.1, dodge.width = 0.9),
               alpha = 0.2, guide = F) +
  geom_violin(colour = "black", lwd = 0.6) +
  facet_wrap(netprop ~., scales = "free_y",
             labeller = as_labeller(c("btw" = "Centralidade", "degree" = "Grau"))) +
  guides(col = F) +
  scale_x_discrete("", labels = c("mouse" = "Camundongo", "yeast" = "Levedura")) +
  scale_y_continuous("") +
  scale_fill_discrete("Essencialidade", labels = c("lethal" = "Essencial", "nonlethal" = "Não-essencial
  stat_compare_means(method = "wilcox.test", label = "p.signif", label.x = 1.5)
```




```
ggsave("fig/btw_dg.png", height = 5, width = 7, dpi = 300)
```

Além disso, a principal característica evolutiva também apresenta perfis distintos nestas duas classes.

```
eg %>%
  ggplot(aes(x = organism, y = ancestry, fill = lethal_nonlethal)) +
    geom_violin() +
    guides(fill = guide_legend(title = "Essencialidade",
                               override.aes = aes(label = "")),
           col = F) +
    scale_x_discrete("", labels = c("mouse" = "Camundongo", "yeast" = "Levedura")) +
    scale_y_continuous("Escore de ancestralidade") +
    scale_fill_discrete("Essencialidade", labels = c("lethal" = "Essencial", "nonlethal" = "Não-essenci
  stat_compare_means(method = "wilcox.test", label = "p.signif", label.x = 1.5)
```

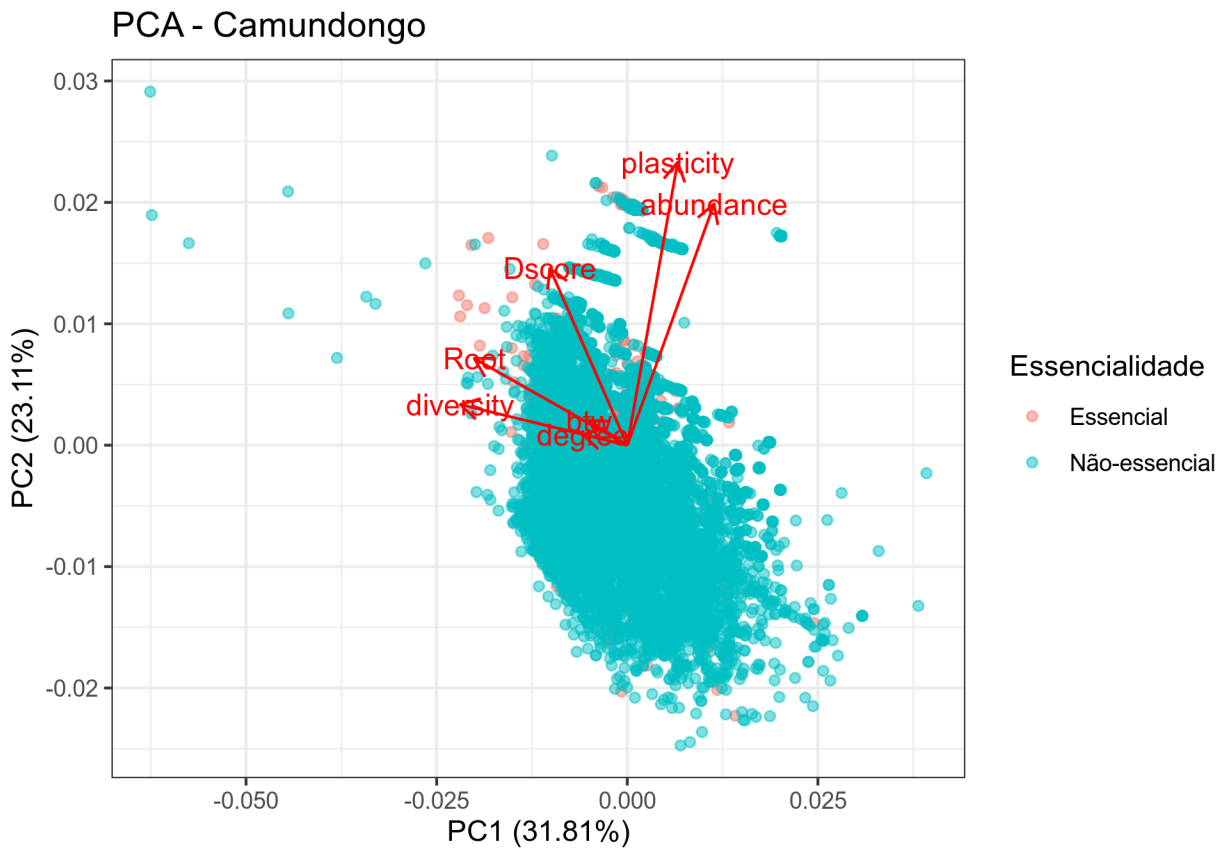



```
ggsave("fig/ancestry.png", height = 5, width = 7, dpi = 300)
```

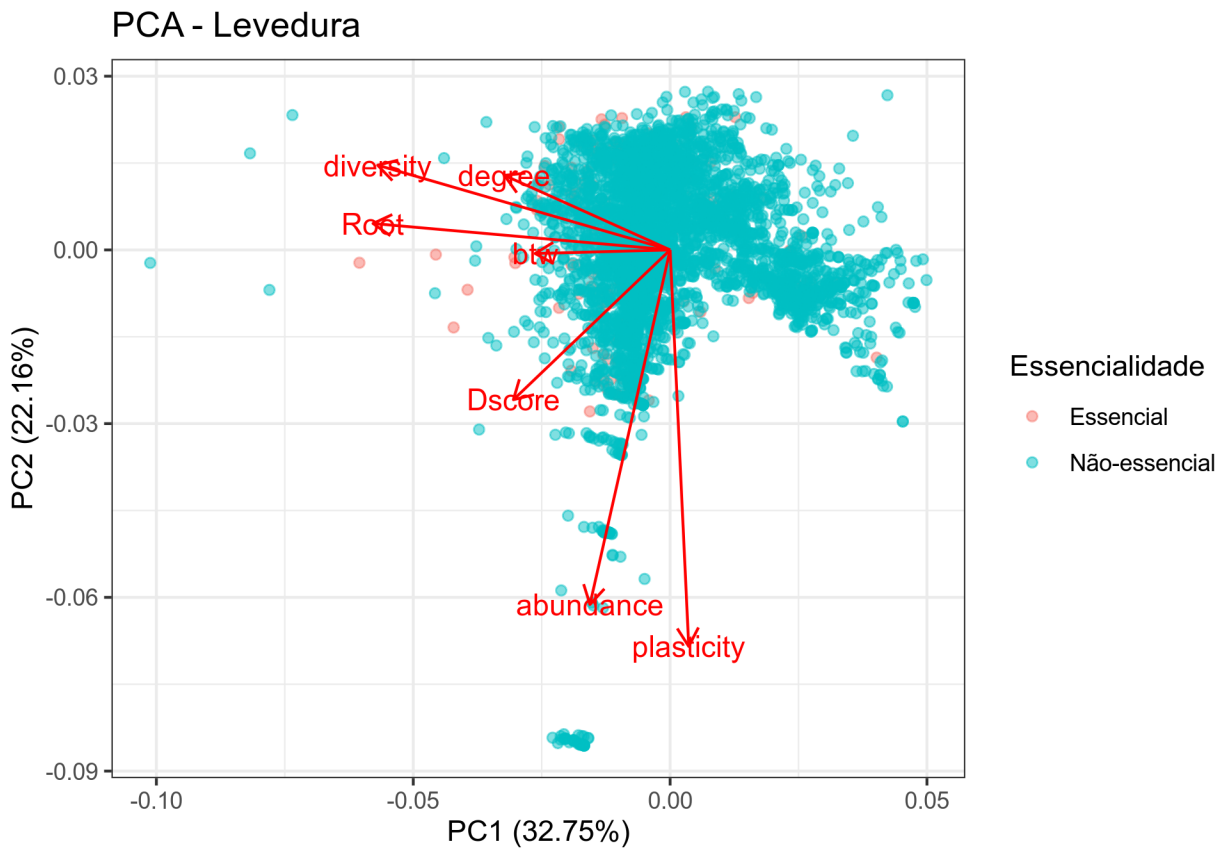
Com estes gráficos, pode-se perceber que o perfil de essencialidade não é claramente separado pelas variáveis centralidade, grau do nó e ancestralidade. Isto pode ainda ser observado com o gráfico das componentes principais, o qual mostra ambos os perfis de essencialidade misturados entre si.

As variáveis `ensembl_peptide_id`, `cog_id`, `Pvalue`, `AdjPvalue` não são necessárias e foram removidas da análise. A variável `ancestry` foi removida pois ela é redundante à variável `Root`, uma vez que aquela é uma transformação desta para a escala 0-1.

```
# Análise exploratória de camundongo ----
mouse <- eg %>%
  # Remover variáveis
  # A variável 'ancestry' é redundante à 'Root'
  select(-c(ensembl_peptide_id, cog_id, Pvalue, AdjPvalue, ancestry)) %>%
  relocate(lethal_nonlethal) %>%
  filter(organism == "mouse") %>%
  select(-organism)

autoplot(prcomp(mouse %>% select(-lethal_nonlethal),
                center = TRUE, scale. = TRUE),
         data = mouse,
         colour = "lethal_nonlethal", alpha = 0.5,
         loadings = TRUE, loadings.label = TRUE) +
  scale_colour_discrete("Essencialidade", labels = c("lethal" = "Essencial", "nonlethal" = "Não-essenci
    ggtitle("PCA - Camundongo")
```




```
ggsave("fig/pca_mouse.png", height = 5, width = 7, dpi = 300)

# Análise exploratória de levedura ----
yeast <- eg %>%
  # Remover variáveis
  # A variável 'ancestry' é redundante à 'Root'
  select(-c(ensembl_peptide_id, cog_id, Pvalue, AdjPvalue, ancestry)) %>%
  relocate(lethal_nonlethal) %>%
  filter(organism == "yeast") %>%
  select(-organism)

autoplot(prcomp(yeast %>% select(-lethal_nonlethal),
                center = TRUE, scale. = TRUE),
         data = yeast,
         colour = "lethal_nonlethal", alpha = 0.5,
         loadings = TRUE, loadings.label = TRUE) +
    scale_colour_discrete("Essencialidade", labels = c("lethal" = "Essencial", "nonlethal" = "Não-essen
  ggtitle("PCA - Levedura")
```

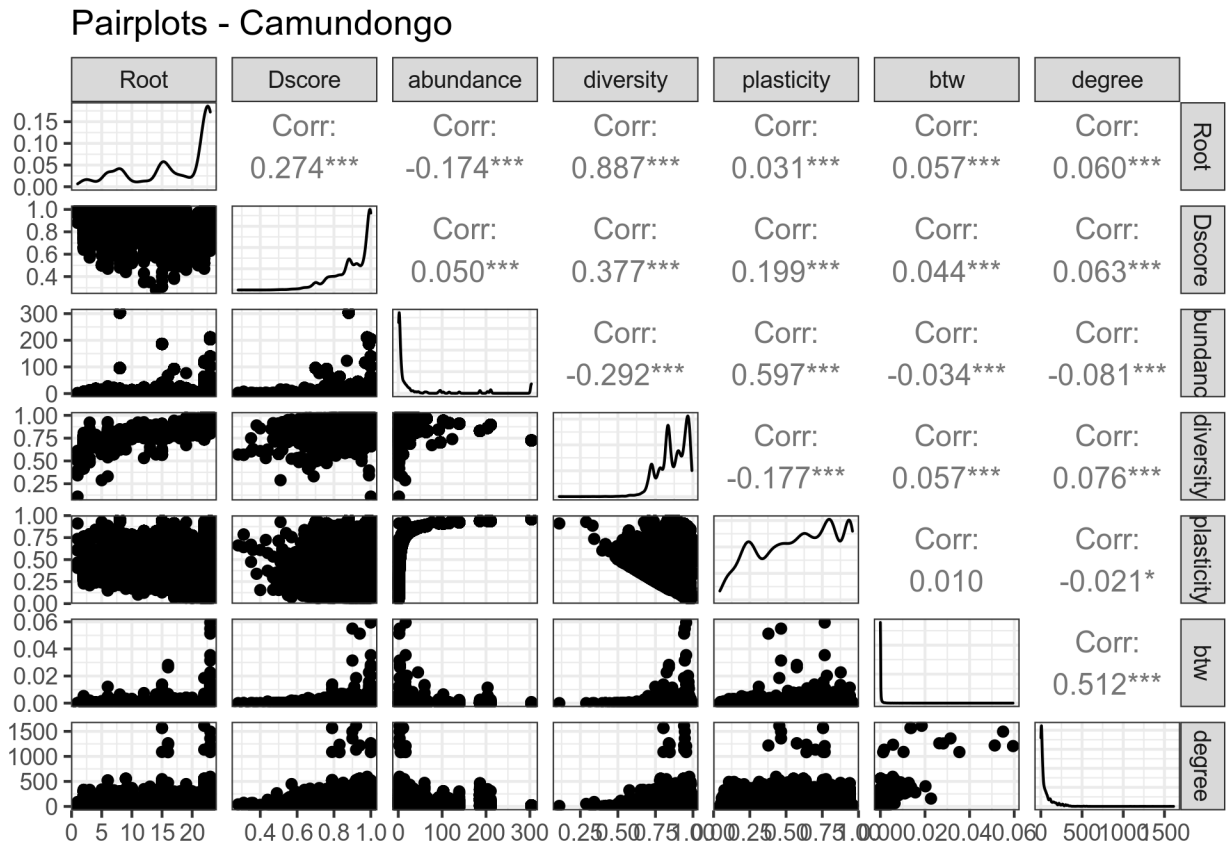



```
ggsave("fig/pca_yeast.png", height = 5, width = 7, dpi = 300)
```

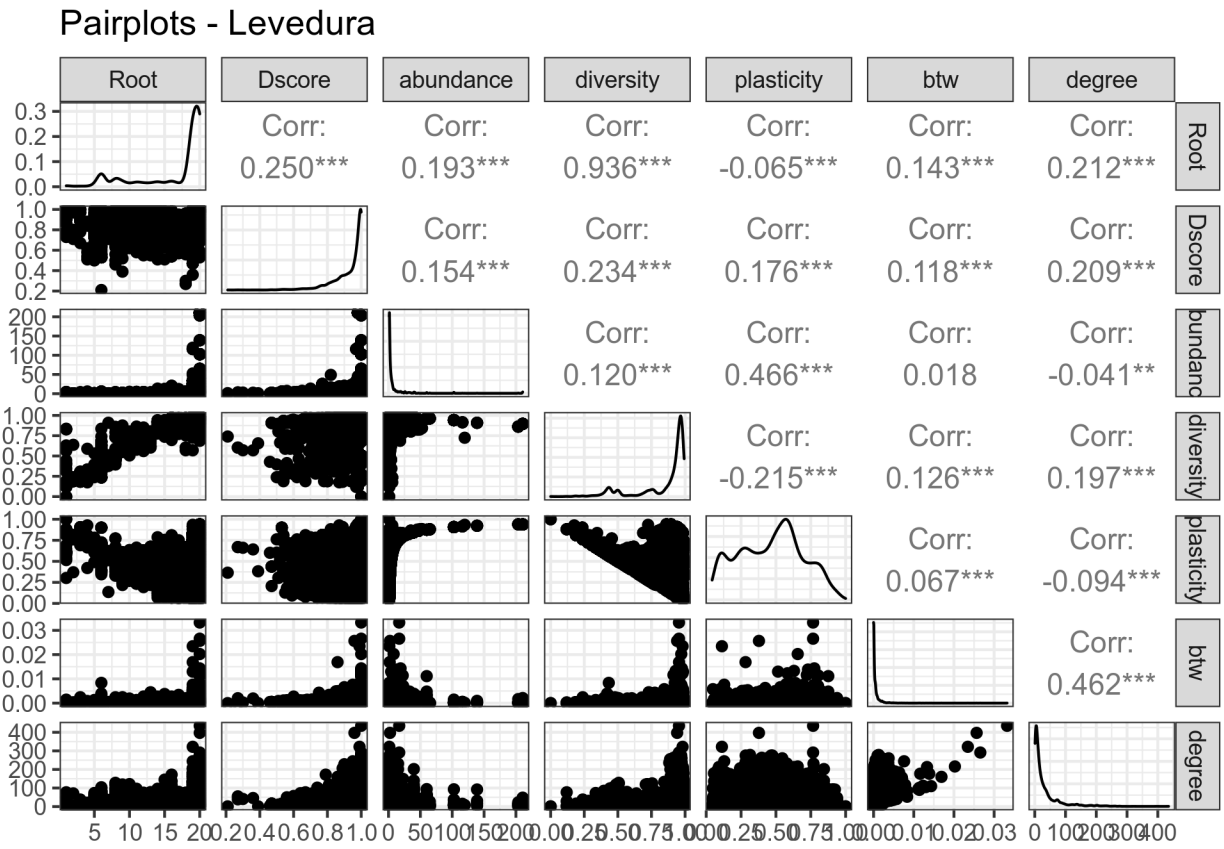
É importante notar que há muito mais genes não-esssenciais anotados que os essenciais em ambos os organismos, sendo necessária a adequação das proporções destes dois grupos nos conjuntos treino e teste dos modelos a serem criados.

A distribuição das variáveis, de uma forma geral, não segue a distribuição normal, como pode ser observado nos gráficos abaixo. Podemos ver também que algumas variáveis são correlacionadas entre si, mas o tipo de relação entre elas (se linear ou exponencial, por exemplo) não está claro. As maiores correlações então entre as variáveis dentro do perfil evolutivo e dentro do perfil das propriedades de redes.

```
mouse %>%
  select(-lethal_nonlethal) %>%
  ggpairs() +
  ggtitle("Pairplots - Camundongo")
```




```
ggsave("fig/pairs_mouse.png", height = 5, width = 7, dpi = 300)

yeast %>%
  select(-lethal_nonlethal) %>%
  ggpairs() +
  ggtitle("Pairplots - Levedura")
```




```
ggsave("fig/pairs_yeast.png", height = 5, width = 7, dpi = 300)
```

Modelos

Para a tarefa de classificação, aplicou-se os algoritmos Random Forest, SVM polinomial e SVM radial para os genes de ambos os organismos. Este algoritmos foram escolhidos pois eles mostraram-se eficientes para tarefas de classificação nas quais não há uma separação linear dos grupos, como mostrado na PCA dos dois organismos. As variáveis numéricas foram centralizadas e escaladas, com nivelamento do número de observações para cada grupo nos conjuntos treino e teste.

Os hiperparâmetros foram os seguintes:

- Random Forest:
  - `mtry`: 10 níveis, de 1 a 10.
  - `min_n`: 5 níveis, de 10 a 50.
- SVM polinomial:
  - `cost`: 10 níveis, de -10 a 10. `-degree`: 2 níveis (1 e 3).
- SVM radial:
  - `cost`: 10 níveis, de -10 a 10.
  - `rbf_sigma`: 5 níveis

Camundongo

Os modelos criados para camundongo estão a seguir.

```
#### RANDOM FOREST ####

# Criar os datasets treino e teste ----
```

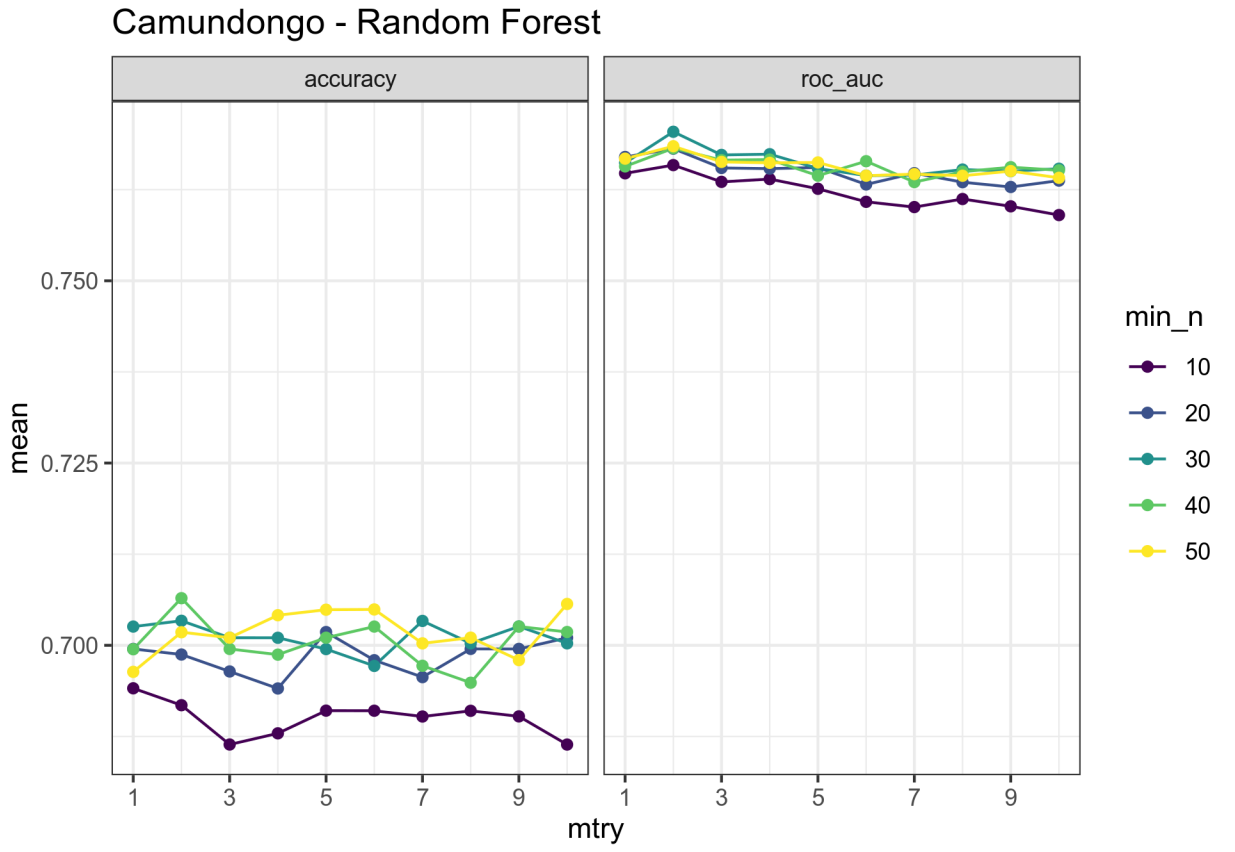
```r
set.seed(1234, kind = "Mersenne-Twister", normal.kind = "Inversion")
mouse_split <- initial_split(mouse, prop = .80, strata = lethal_nonlethal)
mouse_treino <- training(mouse_split)
mouse_teste <- testing(mouse_split)

# Pre-processamento ----
mouse_rec <- recipe(
  lethal_nonlethal ~.,
  data = mouse_treino
) %>%
  themis::step_downsample(lethal_nonlethal) %>%
  step_center(-lethal_nonlethal) %>%
  step_scale(-lethal_nonlethal) %>%
  prep()

mouse_treino_t <- juice(mouse_rec)
mouse_teste_t <- bake(mouse_rec, new_data = mouse_teste)

# Tunning ----
mouse_rf_tune <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

# Grid de procura ----
mouse_rf_grid <- grid_regular(
  mtry(range(1,10)),
  min_n(range(10, 50)),
  levels = c(10, 5)
)

mouse_rf_tune_wflow <- workflow() %>%
  add_model(mouse_rf_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(1111,  kind = "Mersenne-Twister", normal.kind = "Inversion")
mouse_treino_cv <- vfold_cv(mouse_treino_t, v = 10)

mouse_rf_fit_tune <- mouse_rf_tune_wflow %>%
  tune_grid(
    resamples = mouse_treino_cv,
    grid = mouse_rf_grid
  )

#### SVM POLINOMIAL ####

# Tunning
mouse_svm_poly_tune <- svm_poly(
  cost = tune(),
```

```
  degree = tune()
) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

# Grid de procura ----
mouse_svm_poly_grid <- grid_regular(
  cost(range(-10, 10)),
  degree(),
  levels = c(10, 2)
)

mouse_svm_poly_tune_wflow <- workflow() %>%
  add_model(mouse_svm_poly_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(2222,  kind = "Mersenne-Twister", normal.kind = "Inversion")
mouse_treino_cv <- vfold_cv(mouse_treino_t, v = 10)

mouse_svm_poly_fit_tune <- mouse_svm_poly_tune_wflow %>%
  tune_grid(
    resamples = mouse_treino_cv,
    grid = mouse_svm_poly_grid
  )

#### SVM RADIAL ####

# Aproveitar os conjuntos de treino e teste já criados

# Tunning ----
mouse_svm_rbf_tune <- svm_rbf(
  cost = tune(),
  rbf_sigma = tune()
) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

# Grid de procura ----
mouse_svm_rbf_grid <- grid_regular(
  cost(range(-10, 10)),
  rbf_sigma(),
  levels = c(10, 5)
)

mouse_svm_rbf_tune_wflow <- workflow() %>%
  add_model(mouse_svm_rbf_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(3333,  kind = "Mersenne-Twister", normal.kind = "Inversion")
mouse_treino_cv <- vfold_cv(mouse_treino_t, v = 10)
```

```
mouse_svm_rbf_fit_tune <- mouse_svm_rbf_tune_wflow %>%
  tune_grid(
    resamples = mouse_treino_cv,
    grid = mouse_svm_rbf_grid
  )
```

Os modelos foram avaliados e o melhor modelo foi escolhido, com base no critério da acurácia.

```
# Avaliação dos modelos ----

#### RANDOM FOREST ####

# Ver desempenho -----
mouse_rf_fit_tune %>%
  collect_metrics() %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(., aes(x = mtry, y = mean, colour = min_n, group = min_n)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(breaks = seq(1, 9, 2)) +
  scale_colour_viridis_d() +
  ggtitle("Camundongo - Random Forest")
```

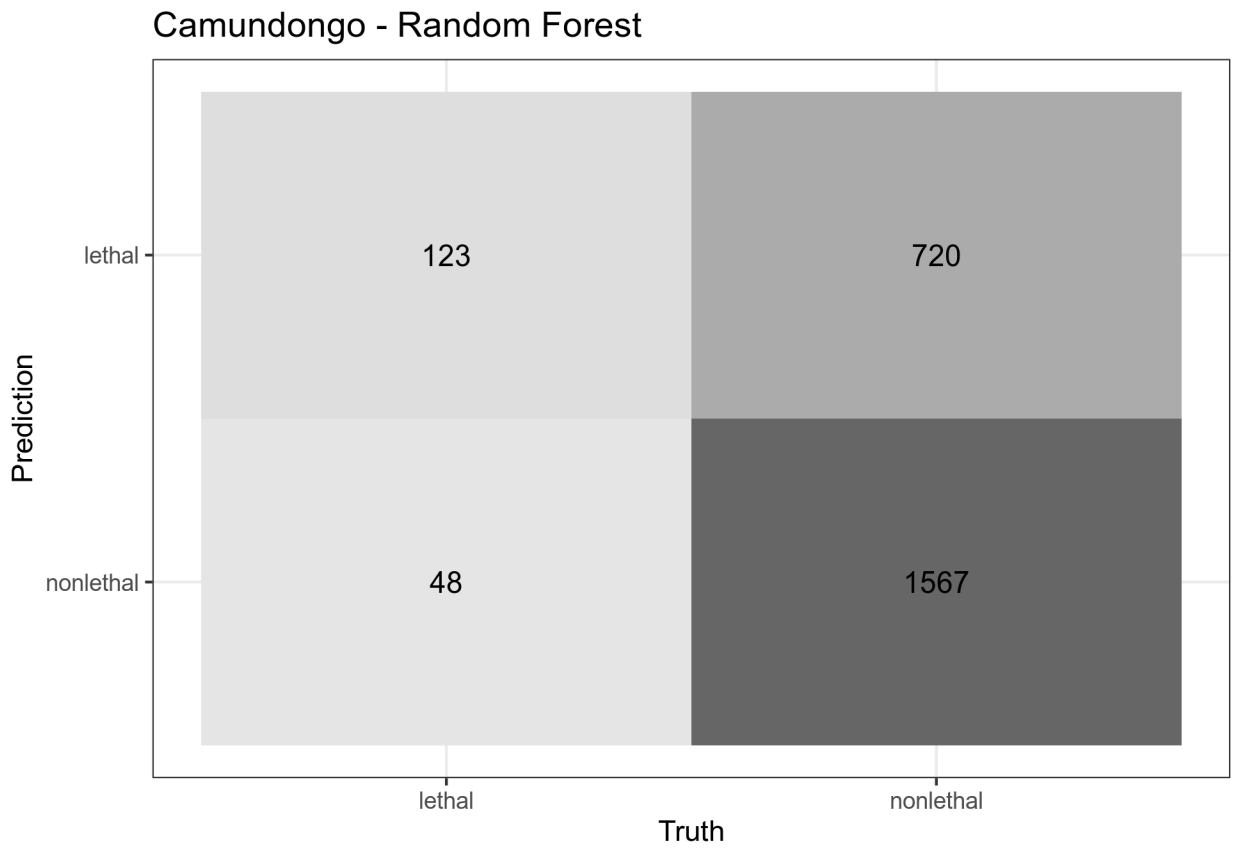



```
ggsave("fig/cv_mouse_rf.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
```

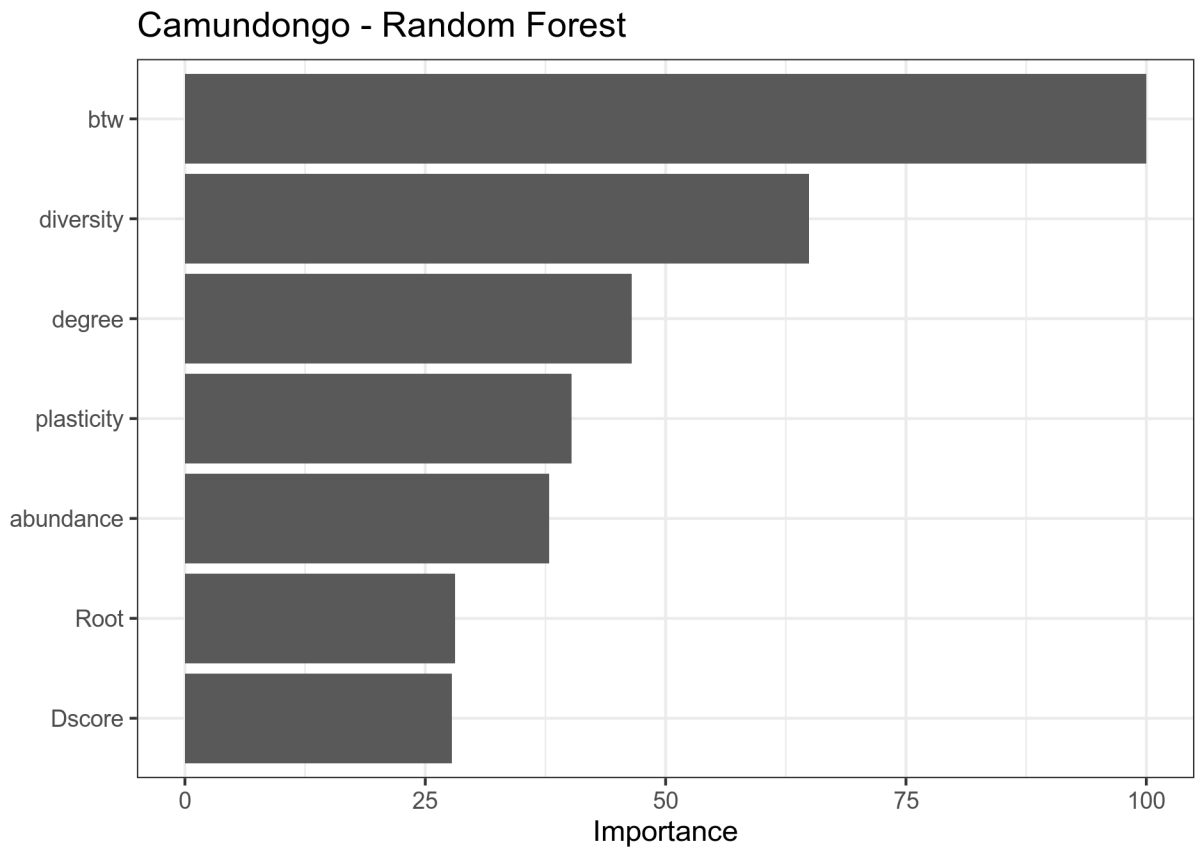
```r
mouse_rf_best <- mouse_rf_fit_tune %>%
  select_best("accuracy")

mouse_rf_final <-
  mouse_rf_tune_wflow %>%
  finalize_workflow(mouse_rf_best)

mouse_rf_final <- fit(mouse_rf_final, mouse_treino_t)

mouse_resultado_rf <-
  mouse_teste_t %>%
  bind_cols(predict(mouse_rf_final, mouse_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(mouse_resultado_rf,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.688
## 2 kap      binary         0.144
```

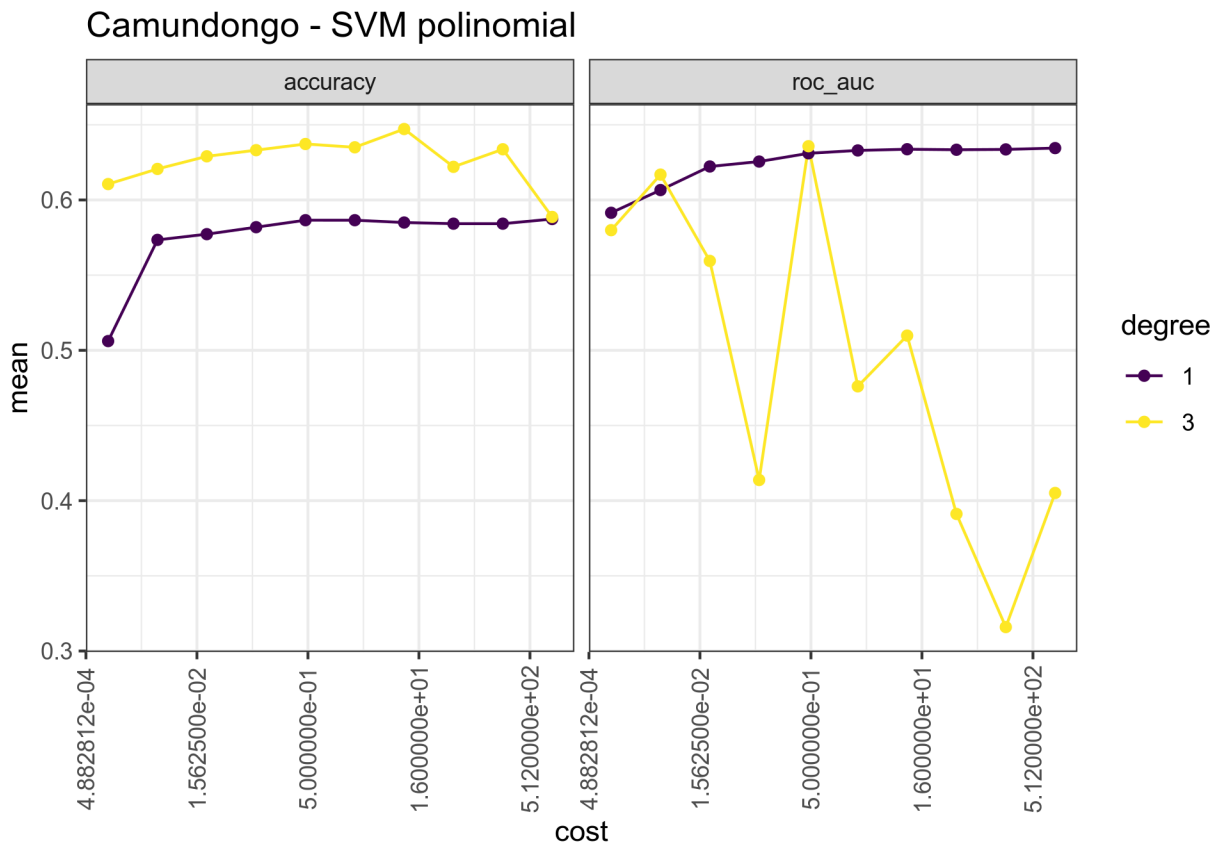
```r
conf_mat(mouse_resultado_rf,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Camundongo - Random Forest")
```




```
ggsave("fig/cm_mouse_rf.png", height = 5, width = 5, dpi = 300)

# Importância das variáveis ----
mouse_rf_final %>%
  pull_workflow_fit() %>%
  vip(scale = TRUE) +
  ggtitle("Camundongo - Random Forest")
```




```
ggsave("fig/rf_importance_mouse.png", height = 5, width = 7, dpi = 300)

#### SVM POLINOMIAL ####

# Ver desempenho ----
mouse_svm_poly_fit_tune %>%
  collect_metrics() %>%
  mutate(degree = factor(degree)) %>%
  ggplot(., aes(x = cost, y = mean, colour = degree)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(trans = "log2") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_colour_viridis_d() +
  ggtitle("Camundongo - SVM polinomial")
```




```
ggsave("fig/cv_mouse_svm_poly.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
mouse_svm_poly_best <- mouse_svm_poly_fit_tune %>%
  select_best("accuracy")

mouse_svm_poly_final <-
  mouse_svm_poly_tune_wflow %>%
  finalize_workflow(mouse_svm_poly_best)

mouse_svm_poly_final <- fit(mouse_svm_poly_final, mouse_treino_t)
```

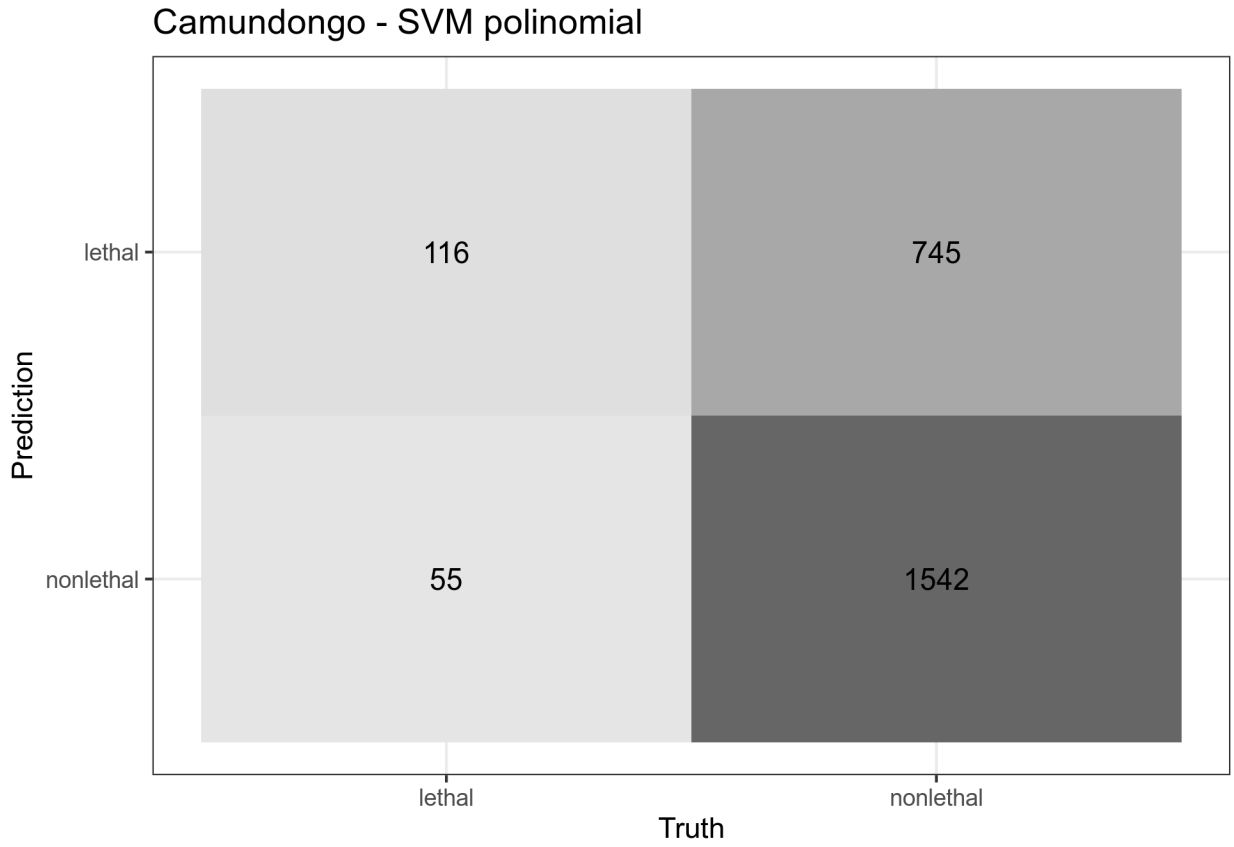
```
## line search fails 9.529659e-05 0.00272536 -0.0003418522 1.419715e-08 6.937368e-12 -3.702351e-12 -2.3
```

```
mouse_resultado_svm_poly <-
  mouse_teste_t %>%
  bind_cols(predict(mouse_svm_poly_final, mouse_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(mouse_resultado_svm_poly,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

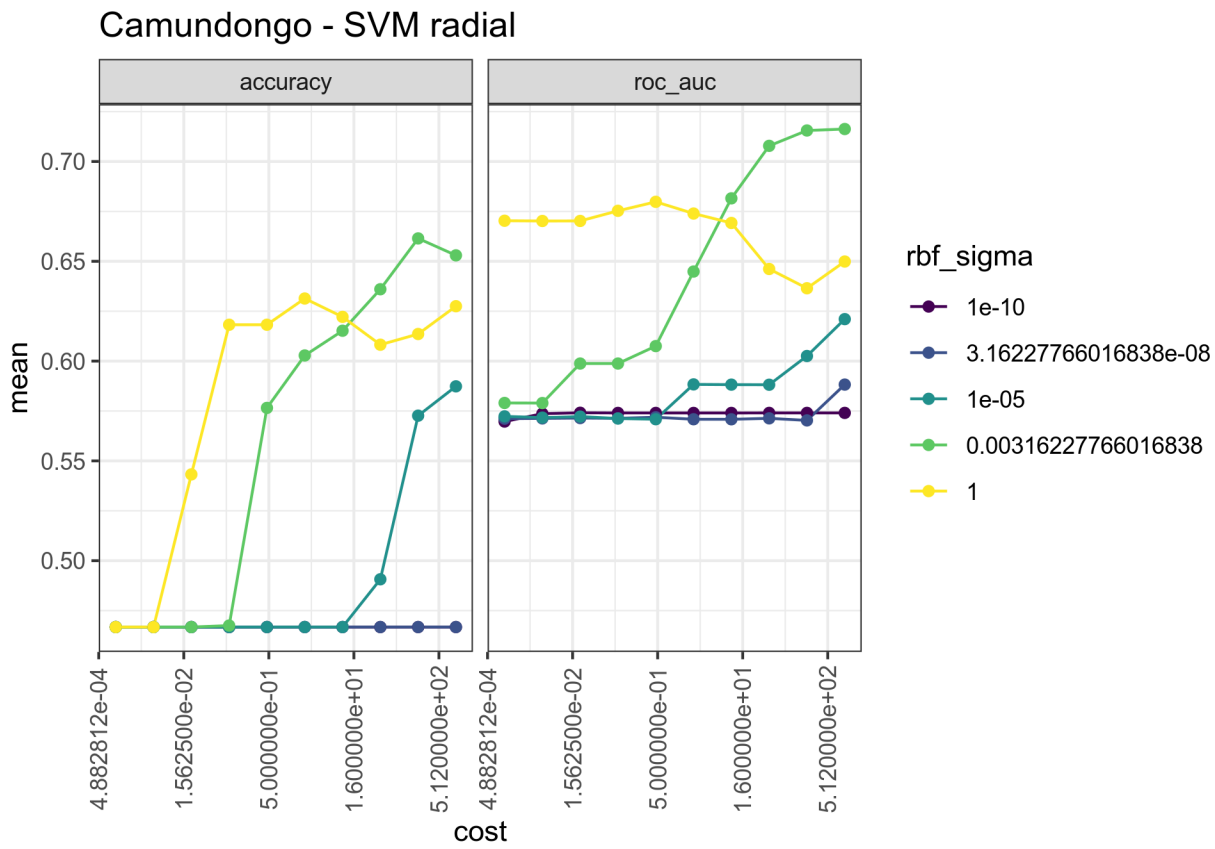
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
```

```
## 1 accuracy binary         0.675
## 2 kap      binary         0.123
```

```
conf_mat(mouse_resultado_svm_poly,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Camundongo - SVM polinomial")
```




```
ggsave("fig/cm_mouse_svm_poly.png", height = 5, width = 5, dpi = 300)

#### SVM RADIAL ####

# Ver desempenho ----
mouse_svm_rbf_fit_tune %>%
  collect_metrics() %>%
  mutate(rbf_sigma = factor(rbf_sigma)) %>%
  ggplot(., aes(x = cost, y = mean, colour = rbf_sigma)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(trans = "log2") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_colour_viridis_d() +
  ggtitle("Camundongo - SVM radial")
```

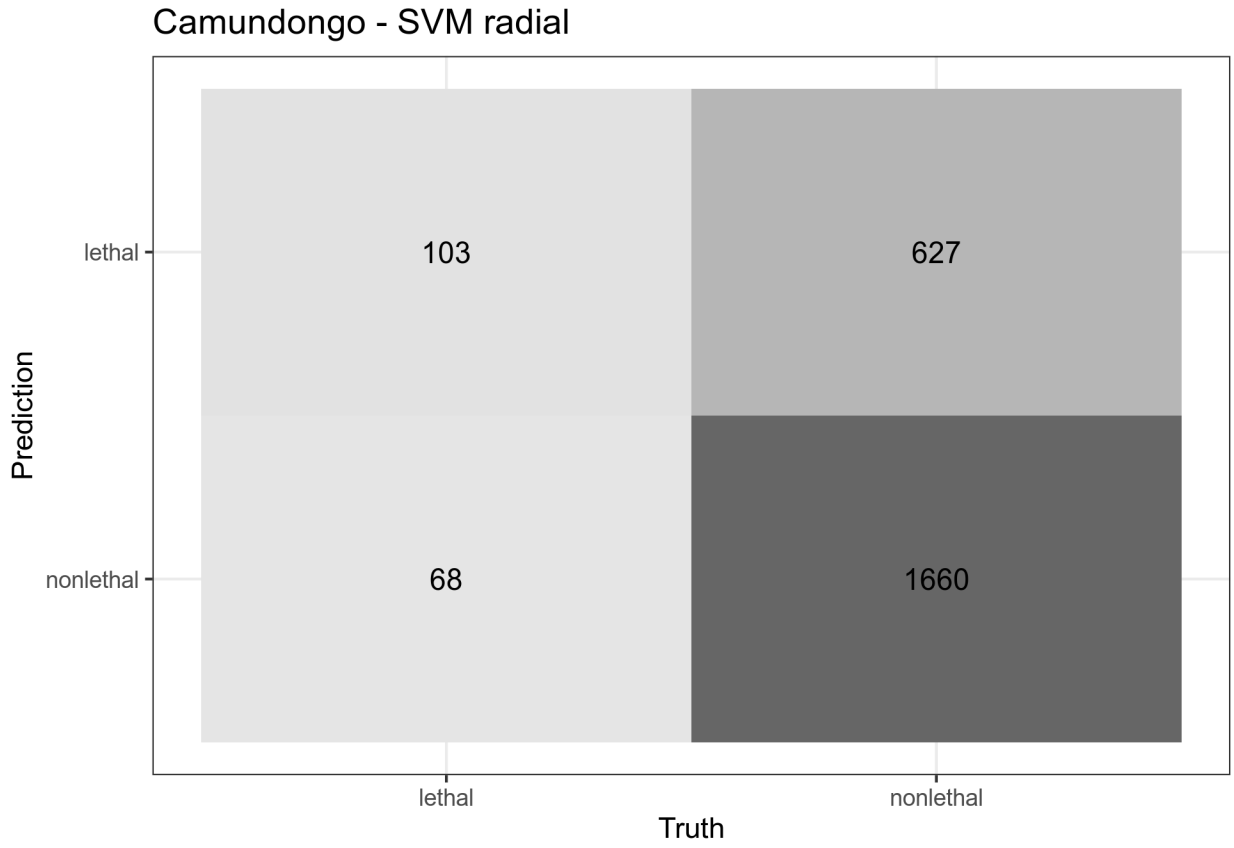



```
ggsave("fig/cv_mouse_svm_rbf.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
mouse_svm_rbf_best <- mouse_svm_rbf_fit_tune %>%
  select_best("accuracy")

mouse_svm_rbf_final <-
  mouse_svm_rbf_tune_wflow %>%
  finalize_workflow(mouse_svm_rbf_best)

mouse_svm_rbf_final <- fit(mouse_svm_rbf_final, mouse_treino_t)

mouse_resultado_svm_rbf <-
  mouse_teste_t %>%
  bind_cols(predict(mouse_svm_rbf_final, mouse_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(mouse_resultado_svm_rbf,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.717
```

```
## 2 kap      binary         0.131
```

```
conf_mat(mouse_resultado_svm_rbf,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Camundongo - SVM radial")
```




```
ggsave("fig/cm_mouse_svm_rbf.png", height = 5, width = 5, dpi = 300)
```

Levedura

Os modelos criados para camundongo estão a seguir.

```
#### RANDOM FOREST ####

# Criar os datasets treino e teste ----
set.seed(1234, kind = "Mersenne-Twister", normal.kind = "Inversion")
yeast_split <- initial_split(yeast, prop = .80, strata = lethal_nonlethal)
yeast_treino <- training(yeast_split)
yeast_teste <- testing(yeast_split)

# Pre-processamento ----
yeast_rec <- recipe(
  lethal_nonlethal ~.,
  data = yeast_treino
) %>%
```

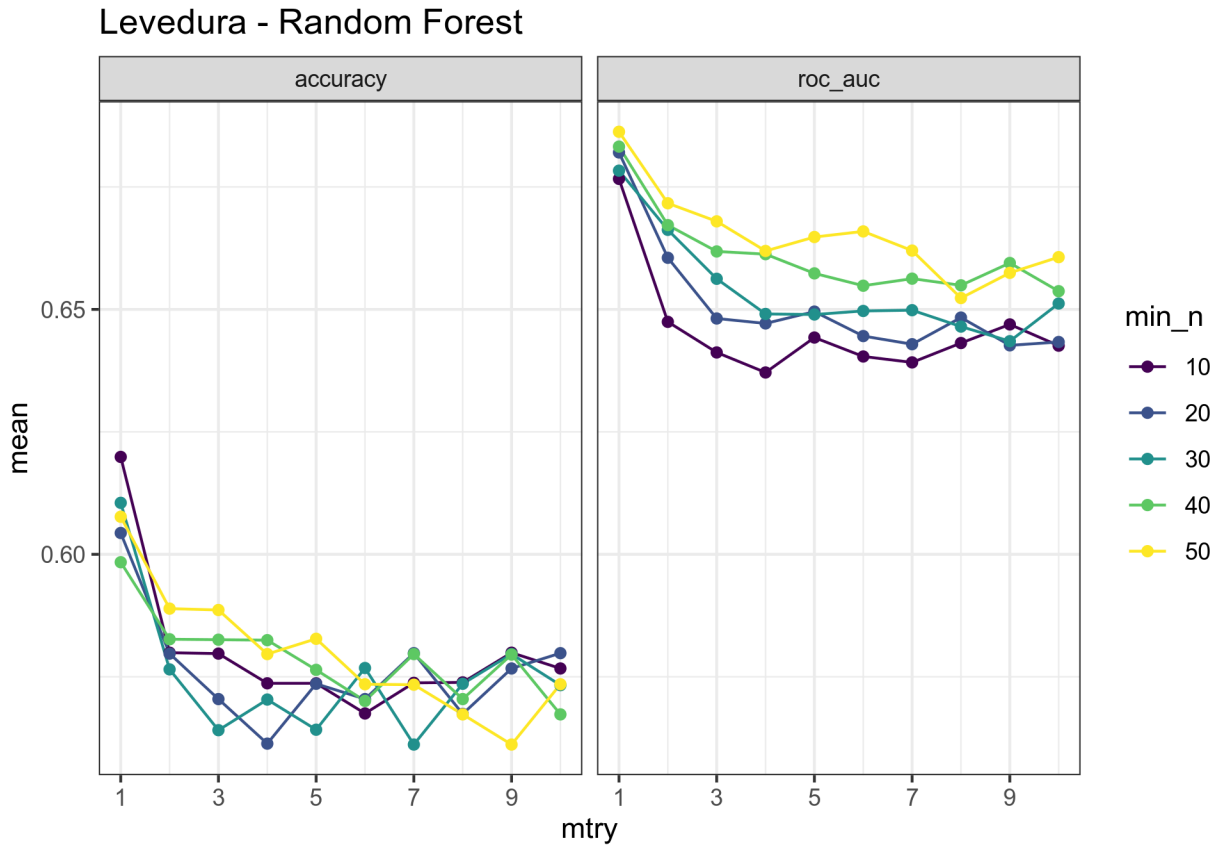
```
  themis::step_downsample(lethal_nonlethal) %>%
  step_center(-lethal_nonlethal) %>%
  step_scale(-lethal_nonlethal) %>%
  prep()

yeast_treino_t <- juice(yeast_rec)
yeast_teste_t <- bake(yeast_rec, new_data = yeast_teste)

# Tunning ----
yeast_rf_tune <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

# Grid de procura ----
yeast_rf_grid <- grid_regular(
  mtry(range(1,10)),
  min_n(range(10, 50)),
  levels = c(10, 5)
)

yeast_rf_tune_wflow <- workflow() %>%
  add_model(yeast_rf_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(1111,  kind = "Mersenne-Twister", normal.kind = "Inversion")
yeast_treino_cv <- vfold_cv(yeast_treino_t, v = 10)

yeast_rf_fit_tune <- yeast_rf_tune_wflow %>%
  tune_grid(
    resamples = yeast_treino_cv,
    grid = yeast_rf_grid
  )

#### SVM POLINOMIAL ####

# Aproveitar os conjuntos de treino e teste já criados

# Tunning
yeast_svm_poly_tune <- svm_poly(
  cost = tune(),
  degree = tune()
) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

yeast_svm_poly_grid <- grid_regular(
  cost(range(-10, 10)),
  degree(),
```

```
  levels = c(10, 2)
)

yeast_svm_poly_tune_wflow <- workflow() %>%
  add_model(yeast_svm_poly_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(2222,  kind = "Mersenne-Twister", normal.kind = "Inversion")
yeast_treino_cv <- vfold_cv(yeast_treino_t, v = 10)

yeast_svm_poly_fit_tune <- yeast_svm_poly_tune_wflow %>%
  tune_grid(
    resamples = yeast_treino_cv,
    grid = yeast_svm_poly_grid
  )

#### SVM RADIAL ####

# Aproveitar os conjuntos de treino e teste já criados

# Tunning
yeast_svm_rbf_tune <- svm_rbf(
  cost = tune(),
  rbf_sigma = tune()
) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

yeast_svm_rbf_grid <- grid_regular(
  cost(range(-10, 10)),
  rbf_sigma(),
  levels = c(10, 5)
)

yeast_svm_rbf_tune_wflow <- workflow() %>%
  add_model(yeast_svm_rbf_tune) %>%
  add_formula(lethal_nonlethal ~.)

# Validação cruzada ----
set.seed(3333,  kind = "Mersenne-Twister", normal.kind = "Inversion")
yeast_treino_cv <- vfold_cv(yeast_treino_t, v = 10)

yeast_svm_rbf_fit_tune <- yeast_svm_rbf_tune_wflow %>%
  tune_grid(
    resamples = yeast_treino_cv,
    grid = yeast_svm_rbf_grid
  )
```

Avaliação dos modelos foi feita a seguir. Os melhores modelos também foram escolhidos com base na acurácia.

```
# Avaliação dos modelos ----

#### RANDOM FOREST ####
```

```r
# Ver desempenho ----
yeast_rf_fit_tune %>%
  collect_metrics() %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(., aes(x = mtry, y = mean, colour = min_n, group = min_n)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(breaks = seq(1, 9, 2)) +
  scale_colour_viridis_d() +
  ggtitle("Levedura - Random Forest")
```

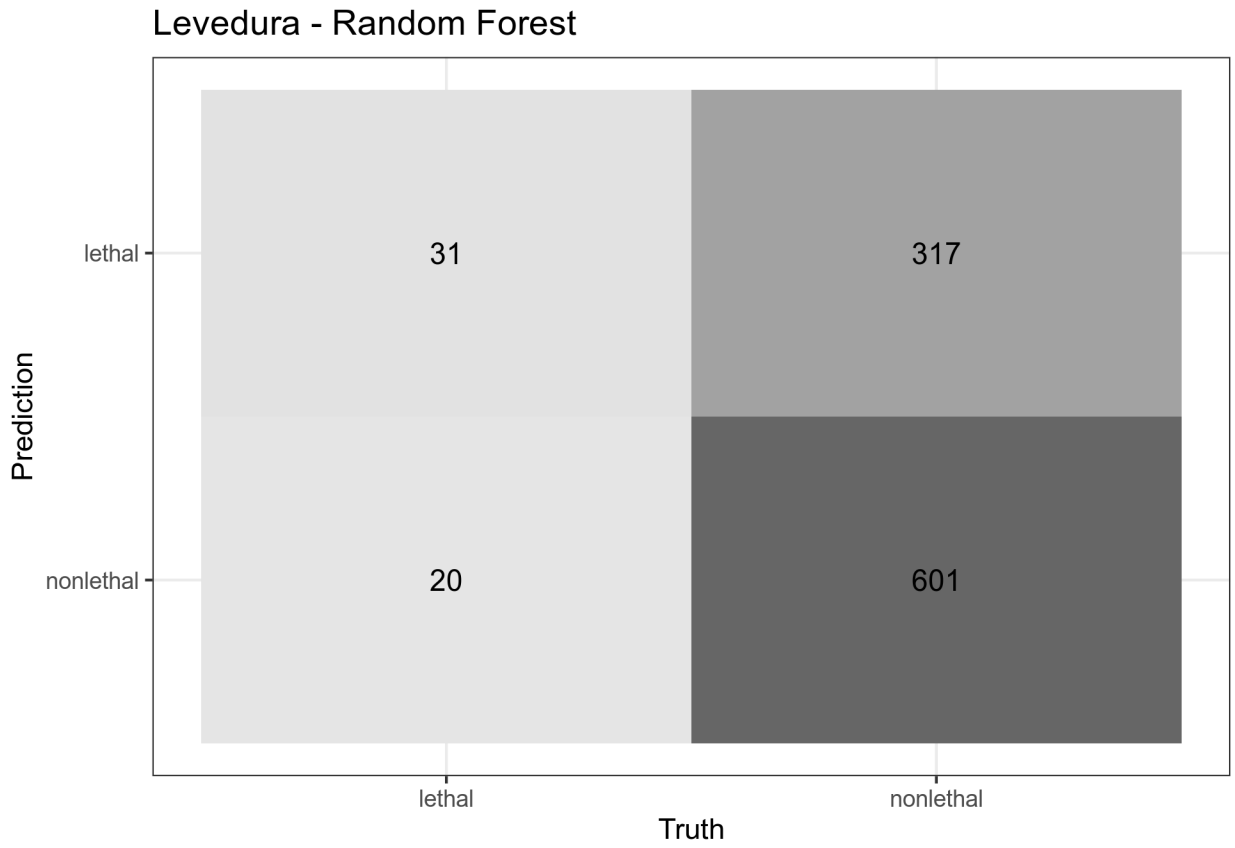



```r
ggsave("fig/cv_yeast_rf.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
yeast_rf_best <- yeast_rf_fit_tune %>%
  select_best("accuracy")

yeast_rf_final <-
  yeast_rf_tune_wflow %>%
  finalize_workflow(yeast_rf_best)

yeast_rf_final <- fit(yeast_rf_final,
                      yeast_treino_t)
```

```
yeast_resultado_rf <-
  yeast_teste_t %>%
  bind_cols(predict(yeast_rf_final, yeast_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(yeast_resultado_rf,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

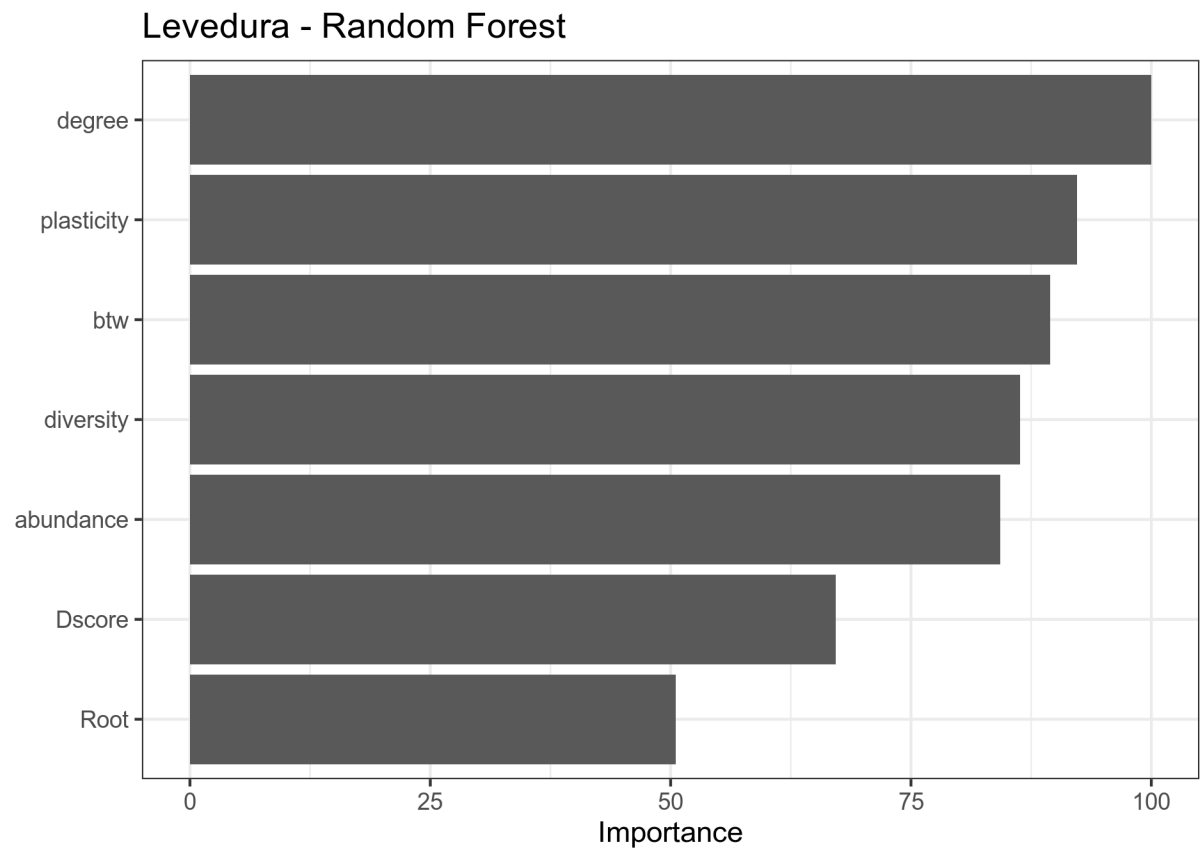
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.652
## 2 kap      binary         0.0700
```

```
conf_mat(yeast_resultado_rf,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Levedura - Random Forest")
```

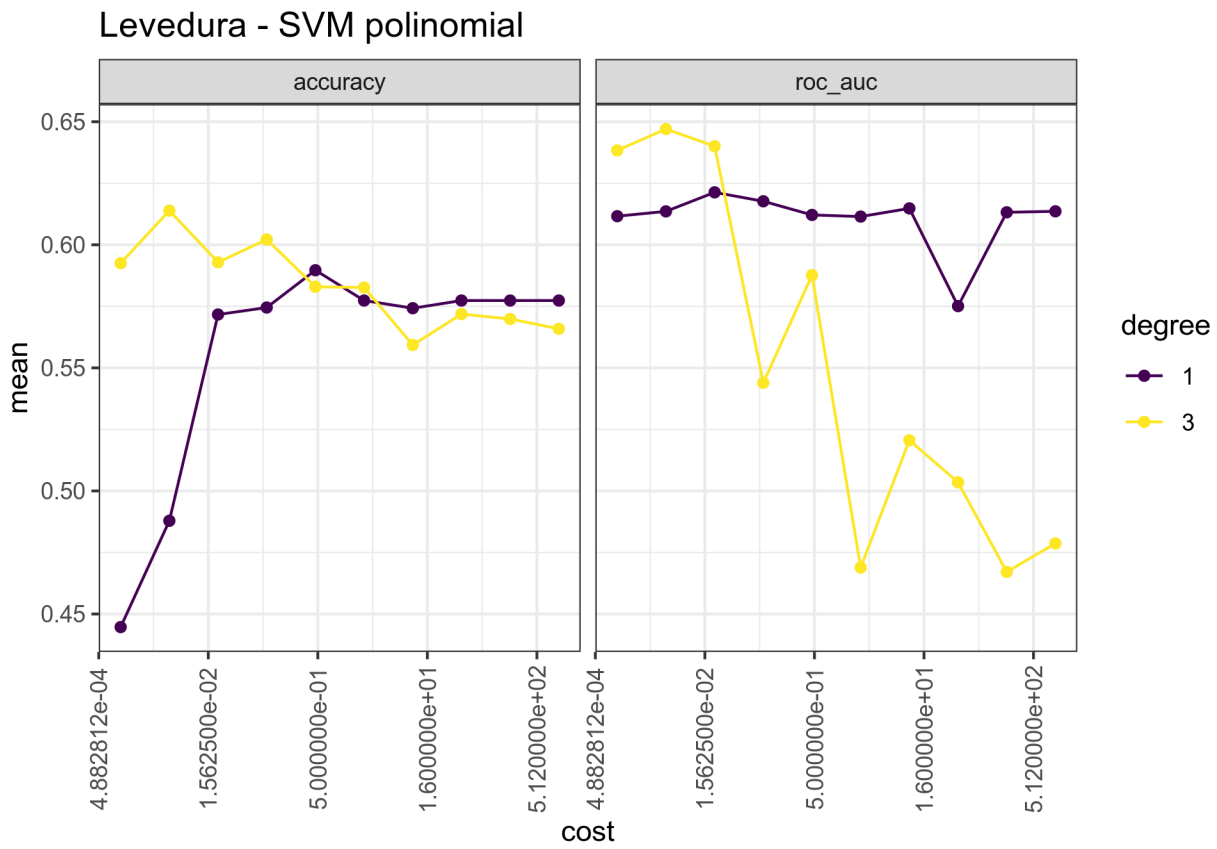



```
ggsave("fig/cm_yeast_rf.png", height = 5, width = 5, dpi = 300)

# Importância das variáveis ----
yeast_rf_final %>%
  pull_workflow_fit() %>%
```

```
  vip(scale = TRUE) +
  ggtitle("Levedura - Random Forest")
```




```
ggsave("fig/rf_importance_yeast.png", height = 5, width = 7, dpi = 300)

#### SVM POLINOMIAL ####

# Ver desempenho ----
yeast_svm_poly_fit_tune %>%
  collect_metrics() %>%
  mutate(degree = factor(degree)) %>%
  ggplot(., aes(x = cost, y = mean, colour = degree)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(trans = "log2") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_colour_viridis_d() +
  ggtitle("Levedura - SVM polinomial")
```

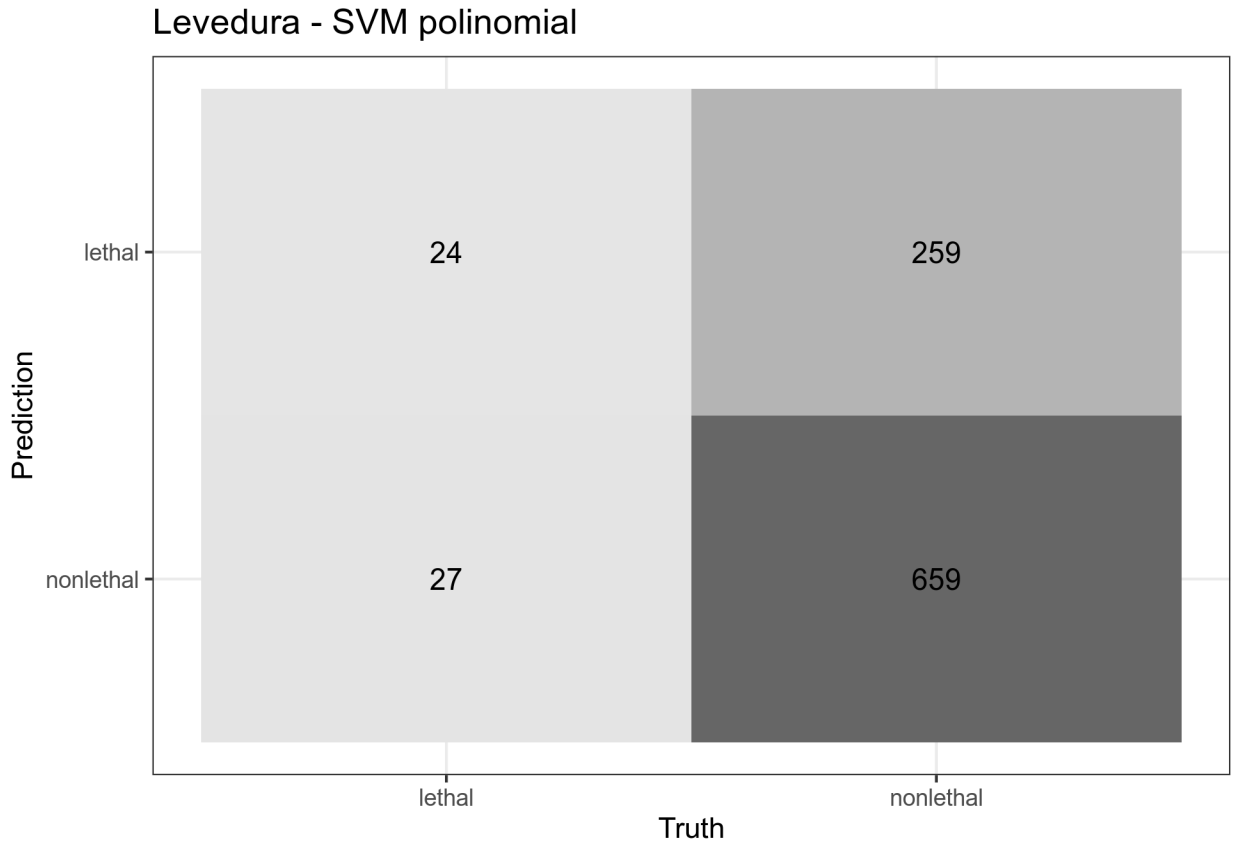



```
ggsave("fig/cv_yeast_svm_poly.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
yeast_svm_poly_best <- yeast_svm_poly_fit_tune %>%
  select_best("accuracy")

yeast_svm_poly_final <-
  yeast_svm_poly_tune_wflow %>%
  finalize_workflow(yeast_svm_poly_best)

yeast_svm_poly_final <- fit(yeast_svm_poly_final, yeast_treino_t)

yeast_resultado_svm_poly <-
  yeast_teste_t %>%
  bind_cols(predict(yeast_svm_poly_final, yeast_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(yeast_resultado_svm_poly,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

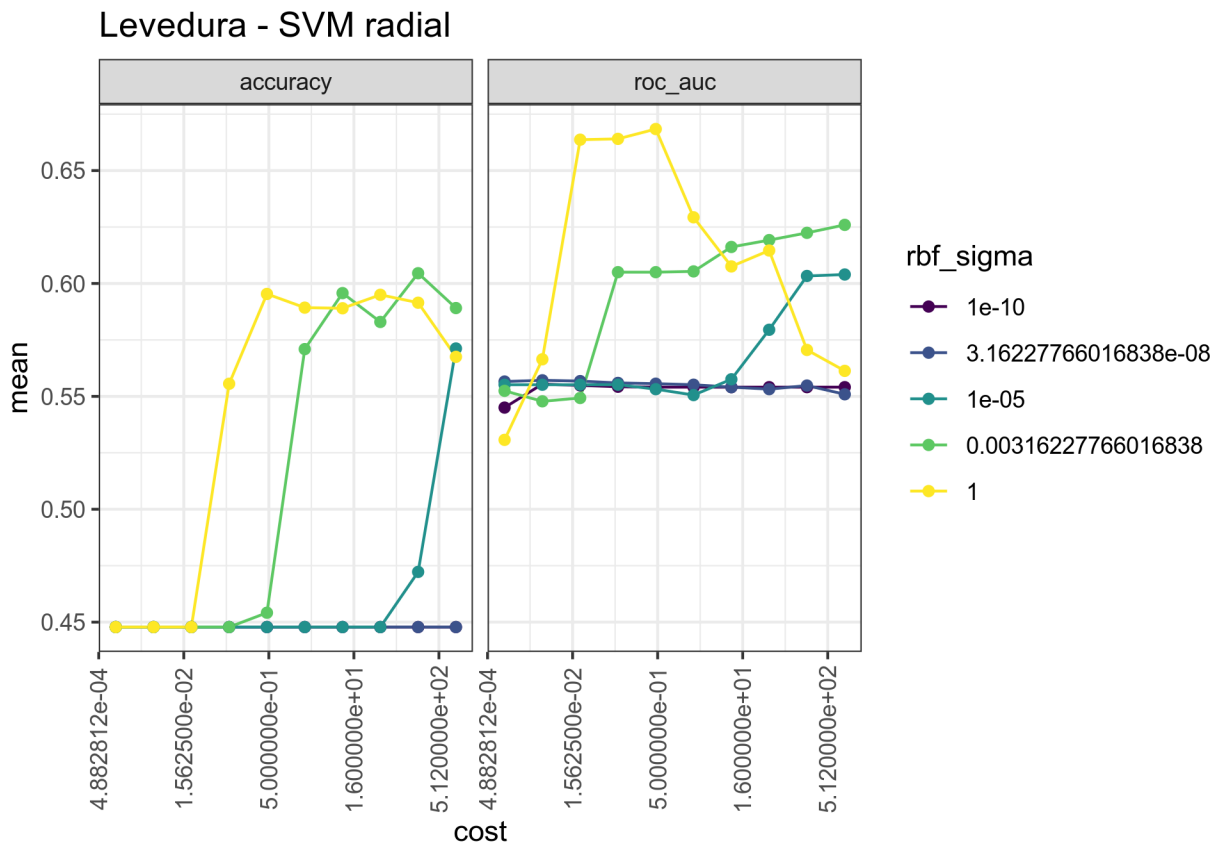
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.705
```

```
## 2 kap      binary         0.0599
```

```
conf_mat(yeast_resultado_svm_poly,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Levedura - SVM polinomial")
```




```
ggsave("fig/cm_yeast_svm_poly.png", height = 5, width = 5, dpi = 300)

#### SVM RADIAL ####

# Ver desempenho ----
yeast_svm_rbf_fit_tune %>%
  collect_metrics() %>%
  mutate(rbf_sigma = factor(rbf_sigma)) %>%
  ggplot(., aes(x = cost, y = mean, colour = rbf_sigma)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(trans = "log2") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_colour_viridis_d() +
  ggtitle("Levedura - SVM radial")
```

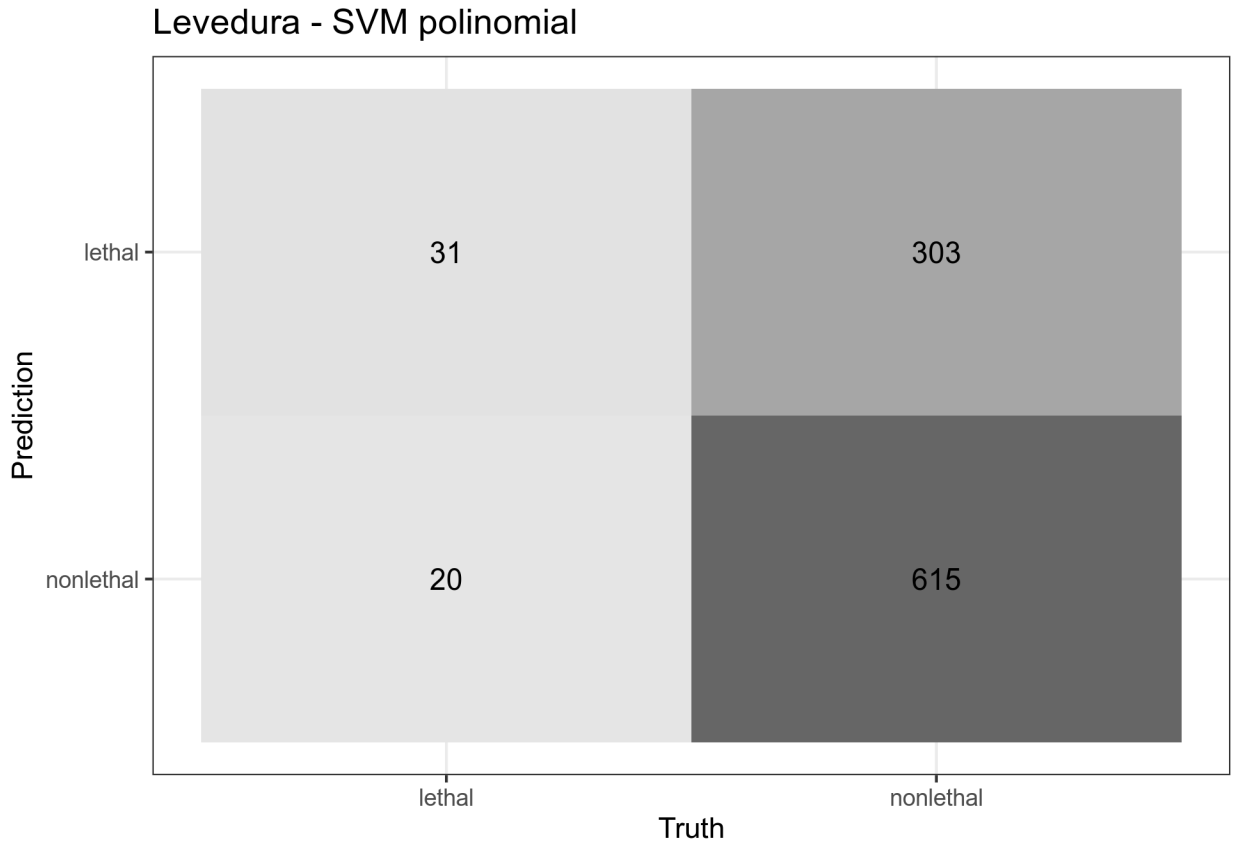



```
ggsave("fig/cv_yeast_svm_rbf.png", height = 5, width = 7, dpi = 300)

# Selecionar melhor modelo ----
yeast_svm_rbf_best <- yeast_svm_rbf_fit_tune %>%
  select_best("accuracy")

yeast_svm_rbf_final <-
  yeast_svm_rbf_tune_wflow %>%
  finalize_workflow(yeast_svm_rbf_best)

yeast_svm_rbf_final <- fit(yeast_svm_rbf_final, yeast_treino_t)

yeast_resultado_svm_rbf <-
  yeast_teste_t %>%
  bind_cols(predict(yeast_svm_rbf_final, yeast_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(yeast_resultado_svm_rbf,
        truth = lethal_nonlethal,
        estimate = predicao_rf,
        options = "roc")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.667
```

```
## 2 kap      binary         0.0767
```

```
conf_mat(yeast_resultado_svm_rbf,
         truth = lethal_nonlethal,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap") +
  ggtitle("Levedura - SVM polinomial")
```




```
ggsave("fig/cm_yeast_svm_rbf.png", height = 5, width = 5, dpi = 300)
```

Os resultados mostram que nenhum dos três algoritmos utilizados conseguiu uma boa acurácia para a predição das classes nos dois organismos, com todos os modelos apresentando acurácia em torno de 70%. Entretanto, o modelo com melhor desempenho em camundongo foi o Random Forest, com os hiperparâmetros `mtry = 1` e `min_n = 10`, resultando em 69.9% de acurácia média no conjunto de treino. Para levedura, o melhor modelo foi o o SVM polinomial, com acurácia média de 70,5% no conjunto de treino.

Em todos os modelos, observa-se uma alta taxa de falsos-positivos, representados na matriz de confusão pelo número de genes não-essenciais preditos como essenciais. Tal comportamento pode ser explicado pela grande variabilidade de características dos genes não-essenciais. Em outras palavras, em termos de características de redes biológicas e evolutivas, os genes essenciais são mais homogêneos que os não essenciais. Além disso, como mostram os gráficos da importância das variáveis estimadas nos modelos de Random Forest, cada organismo possui características particulares as quais definem a essencialidade de seus genes.

Para a obtenção de modelos melhores, duas estratégias podem ser utilizadas. A primeira é a coleta de outras variáveis que estejam envolvidas com a essencialidade gênica, para melhorar a separação de entre as duas classes. A segunda consiste em definir melhor um perfil dos genes essenciais e não-essenciais para cada um dos organismos. Entretanto, esta tarefa é árdua, pois a essencialidade de um gene pode estar condicionada a diversos fatores.