

monitorsom: pacote de R para detecção de sinais usando template matching

Roteiro para a oficina do I Simpósio de Física aplicada à Ecologia e Conservação

Gabriel L. M. Rosa

2025-11-05

Contents

1.	Instalação e configuração inicial	2
1.1.	Baixar e instalar o R	2
1.2.	Baixar e instalar o Rstudio	3
1.3.	Instalar o Rtools (passo necessário apenas para usuários do Windows)	3
1.4.	Criar uma nova pasta para conter os arquivos usados pelo monitorsom	3
1.5.	Instalar o devtools	3
1.6.	Carregar o devtools	3
1.7.	Instalar o monitorsom	3
1.8.	Carregar o monitorsom no ambiente de trabalho do R para testar a instalação	3
1.9.	Salvar o script	4
1.10.	Checkpoint 01 - Instalação do monitorsom	4
2.	Organizando o ambiente de trabalho	4
2.1.	Verificar e ajustar o diretório de trabalho	4
2.2.	Carregar arquivos de exemplo	4
2.3.	(Opcional) Limpar arquivos de exemplo	4
2.4.	Checkpoint 02 - Povoamento do diretório de trabalho com os arquivos de exemplo	5
3.	Preparação de templates e soundscapes	5
3.1.	Selecionar templates usando o app de segmentação	5
3.2.	Segmentar as soundscapes	5
3.3.	Preparar os templates	6
3.4.	Exportar os cortes de áudio dos templates	6
3.5.	Checkpoint 03 - Decisão sobre o método de processamento	7
4.	Obtendo as detecções	7
4.1.	Processo simplificado	7

4.2. Processo detalhado: metadados das soundscapes	8
4.3. Processo detalhado: metadados dos templates	9
4.4. Processo detalhado: criar grade de busca	10
4.5. Processo detalhado: executar template matching	11
4.6. Caso especial: obter scores brutos	11
4.7. Visualizar os scores brutos	12
4.8. Checkpoint 04 - Filtragem adicional de detecções	19
5. Validação das detecções	19
5.1. Validação a priori	19
5.2. Inspecionar resultados da validação	20
5.3. Validação a posteriori (manual)	21
5.4. Importar resultados da validação a posteriori	22
5.5. Comparar validações a priori e a posteriori	22
6. Diagnósticos de performance e resultados finais	22
6.1. Diagnósticos de performance	22
6.2. Obter conjunto final de detecções	24
6.2.1. Extrair limiares e nomes dos templates	24
6.2.2. Filtrar detecções do template 1	25
6.2.3. Filtrar detecções do template 2	25
6.3. Converter detecções finais em tabelas de ROIs	26
Considerações finais	26
Resumo do processo	26
Adaptação para diferentes projetos	27
Próximos passos e extensões	27
Boas práticas recomendadas	27
Suporte e contribuições	27

1. Instalação e configuração inicial

Seja bem-vindo(a) ao tutorial do monitoraSom! Nessa primeira etapa, vamos fazer os downloads e instalações necessárias para o uso do monitoraSom.

1.1. Baixar e instalar o R

- Verifique se já existe alguma versão instalada.
- Ao fazer uma nova instalação, dê preferência para a versão mais recente.
- link para download: <https://cran.r-project.org/bin/windows/base/>

1.2. Baixar e instalar o Rstudio

- Após instalar o R, instale o Rstudio.
- Atenção: se houver mais de uma versão do R instalada, certifique-se qual está sendo usada ao abrir o Rstudio.
- link para download: <https://posit.co/download/rstudio-desktop/>

1.3. Instalar o Rtools (passo necessário apenas para usuários do Windows)

- Rtools é necessário para compilar pacotes do R que foram instalados a partir do GitHub.
- Verifique se já existe alguma versão instalada. Caso exista, verifique se é adequada ao projeto. Se não for, desinstale a versão existente.
- Neste momento, a versão mais recente é a 4.5.
- link para download: <https://cran.r-project.org/bin/windows/Rtools/>

1.4. Criar uma nova pasta para conter os arquivos usados pelo monitoraSom

Recomenda-se criar uma pasta para análises na raiz do sistema (ex: “C:/Meus Projetos/monitoraSom_tutorial”), garantindo que os caminhos permaneçam consistentes mesmo ao mover arquivos entre computadores.

1.5. Instalar o devtools

Instala o pacote `devtools`, necessário para instalar pacotes diretamente do GitHub. Execute no RStudio após instalar o Rtools (apenas Windows).

```
install.packages("devtools")
```

1.6. Carregar o devtools

```
library(devtools)
```

1.7. Instalar o monitoraSom

Instala o pacote `monitoraSom` e suas dependências diretamente do GitHub. Durante a instalação:
- Se solicitado, responda ‘1’ para instalar todas as dependências (“All”)
- Se solicitado sobre compilação, responda ‘Yes’

```
install_github("ConservaSom/monitoraSom", dependencies = TRUE)
```

1.8. Carregar o monitoraSom no ambiente de trabalho do R para testar a instalação

```
library(monitoraSom)
library(dplyr)
library(ggplot2)
library(patchwork)
```

1.9. Salvar o script

Salve o script no diretório de trabalho (passo 1.4) com nome descritivo, por exemplo: “script_01_instalação.R”.

1.10. Checkpoint 01 - Instalação do monitoraSom

Você instalou o monitoraSom e está pronto para começar a usar. Salve esse script no diretório de trabalho conforme descrito no passo 1.4.

2. Organizando o ambiente de trabalho

Vamos seguir a partir deste ponto assumindo que o diretório de trabalho está organizado conforme descrito no passo 1.4 e que o código está salvo em um arquivo em formato de script ou notebook (Rmarkdown ou Quarto).

2.1. Verificar e ajustar o diretório de trabalho

Verifica o diretório de trabalho atual. Se necessário, ajuste usando um dos métodos abaixo.

```
getwd()
```

```
## [1] "/home/grosa/R_repos/2025_SimposioFisicaEcologia_OficinaSomR/monitoraSom"
```

Método 1 - Automático (recomendado): Define o diretório automaticamente baseado na localização do script.

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
getwd()
```

Método 2 - Manual: Define o caminho manualmente para o diretório do projeto.

```
project_path <- paste0(
  "<coloque aqui o caminho para a pasta do projeto>", "/monitoraSom"
)
setwd(project_path)
getwd()
```

2.2. Carregar arquivos de exemplo

Cria a estrutura de diretórios e carrega arquivos de exemplo necessários para o tutorial. **Atenção:** Execute apenas em projetos novos, pois pode sobreescriver dados existentes.

```
set_workspace(project_path = "./", example_data = TRUE)
```

2.3. (Opcional) Limpar arquivos de exemplo

Remove os arquivos de exemplo para iniciar um projeto do zero. Pule esta etapa se desejar usar os dados de exemplo originais.

```

file.remove(list.files("./detections/", full.names = TRUE))
file.remove(list.files("./match_grid_metadata/", full.names = TRUE))
file.remove(list.files("./match_scores/", full.names = TRUE))
file.remove(list.files("./soundscapes_metadata/", full.names = TRUE))
file.remove(list.files("./templates/", full.names = TRUE))
file.remove(list.files("./templates_metadata/", full.names = TRUE))
file.remove(list.files("./validation_outputs/", full.names = TRUE))

```

2.4. Checkpoint 02 - Povoamento do diretório de trabalho com os arquivos de exemplo

Você povoou o diretório de trabalho com os arquivos de exemplo do monitoraSom e está pronto para seguir com o uso do pacote. Apague os arquivos de exemplo se quiser começar um novo projeto.

3. Preparação de templates e soundscapes

3.1. Selecionar templates usando o app de segmentação

Abre o app de segmentação para marcar as regiões de interesse (ROIs) que servirão como templates. Os templates são amostras de som usadas para detecção automática. Neste passo, definimos `cuts_path = "./templates/"` para indicar onde os cortes serão exportados. Os templates serão exportados automaticamente no passo 3.4.

Nota: No exemplo, os ROIs já estão criados. Para criar um novo template, marque um canto completo e adicione o comentário “Complete Song”.

```

launch_segmentation_app(
  user = "User", # Nome do usuário
  project_path = "./", # Caminho para a pasta do projeto
  preset_path = "./app_presets/", # Caminho para a pasta de presets
  soundscapes_path = "./recordings/", # Caminho onde ler as gravações
  roi_tables_path = "./roi_tables/", # Caminho para onde exportar as ROIs
  cuts_path = "./templates/", # Caminho para onde exportar cortes de audio
  dyn_range = c(-102, -42), # Ajuste do contraste do espectrograma
  wl = 1024, # Ajuste do comprimento da janela do fft (parametro espectral)
  ovlp = 50, # Ajuste da sobreposição do fft (parametro espectral)
  color_scale = "greyscale 1", # Ajuste da escala de cores do espectrograma
  nav_autosave = TRUE # Automação de salvamento ao trocar de soundscape
)

```

3.2. Segmentar as soundscapes

Abre o app de segmentação para marcar ROIs nas soundscapes. As tabelas de ROIs geradas serão usadas para validar as detecções e avaliar o desempenho dos templates. Os cortes de áudio são salvos em `./roi_cuts/`.
Nota: No exemplo, a segmentação já está completa; não é necessário marcar novas ROIs.

```

launch_segmentation_app(
  user = "User", project_path = "./", preset_path = "./app_presets/",
  soundscapes_path = "./soundscapes/", roi_tables_path = "./roi_tables/",
  dyn_range = c(-102, -42), wl = 1024, ovlp = 50,
  color_scale = "greyscale 1", visible_bp = TRUE, nav_autosave = TRUE
)

```

3.3. Preparar os templates

Importa todas as tabelas de ROIs e filtra apenas aquelas marcadas como templates. As ROIs de templates e soundscapes estão na mesma pasta, mas podem ser diferenciadas pelo nome do arquivo ou comentário.

```
df_rois <- fetch_rois(rois_path = "./roi_tables/")
unique(df_rois$soundscape_file) # Verificando os nomes das gravações

## [1] "Bcu_1.wav"                      "Bcu_2.wav"
## [3] "W54393S25597_20201104_170000.wav" "W54431S25613_20191102_055000.wav"
## [5] "W54431S25613_20191102_064000.wav" "W54431S25613_20191104_154000.wav"
## [7] "W54431S25613_20191105_060000.wav" "W54443S25620_20191105_054000.wav"
## [9] "W54448S25622_20191101_073000.wav" "W54448S25622_20191101_125000.wav"
## [11] "W54448S25622_20191102_070000.wav" "W54448S25622_20191103_055000.wav"
## [13] "W54448S25622_20191104_171000.wav" "W54448S25623_20191102_064000.wav"
```

Filtrar as ROIs para manter apenas os templates desejados (uma amostra de cada um dos tipos de interesse).

```
df_templates <- df_rois %>%
  filter(roi_comment %in% c("Substructure C", "Complete Song")) %>%
  group_by(roi_comment) %>%
  sample_n(1)
glimpse(df_templates)

## Rows: 2
## Columns: 19
## Groups: roi_comment [2]
## $ soundscape_path      <chr> "./recordings/Bcu_1.wav", "./recordings/Bcu_2.wav"
## $ soundscape_file       <chr> "Bcu_1.wav", "Bcu_2.wav"
## $ roi_path              <chr> "/home/grosa/R_repos/2025_SimposioFisicaEcologia_~"
## $ roi_file               <chr> "Bcu_1_roi_User_20250226101835.csv", "Bcu_2_roi_U~"
## $ roi_user               <chr> "User", "User"
## $ roi_input_timestamp    <chr> "2025-11-05 10:22:43", "26/02/2025 11:12"
## $ roi_label              <chr> "Basileuterus culicivorus", "Basileuterus culiciv~"
## $ roi_start              <dbl> 26.07112, 24.07114
## $ roi_end                <dbl> 28.05989, 24.46161
## $ roi_min_freq           <dbl> 2.732697, 2.877634
## $ roi_max_freq           <dbl> 9.692551, 9.635566
## $ roi_type               <chr> "bird - song", "bird - song"
## $ roi_label_confidence   <chr> "certain", "certain"
## $ roi_is_complete         <chr> "complete", "complete"
## $ roi_comment             <chr> "Complete Song", "Substructure C"
## $ roi_wl                  <int> 1024, 1024
## $ roi_ovlp                 <int> 50, 70
## $ roi_sample_rate          <int> 24000, 24000
## $ roi_pitch_shift          <int> 1, 1
```

3.4. Exportar os cortes de áudio dos templates

Extrai e salva os cortes de áudio das ROIs marcadas como templates na pasta ./templates/.

```

export_roi_cuts(
  df_rois = df_templates, roi_cuts_path = "./templates/", overwrite = FALSE
)
list.files(path = "./templates/", pattern = "Bcu", full.names = TRUE)

```

3.5. Checkpoint 03 - Decisão sobre o método de processamento

Aqui você deve decidir: se deseja seguir com o processo simplificado (função `template_matching()`) execute o passo 4.1, ou se deseja seguir com o processo detalhado, execute os passos 4.2-4.5.

4. Obtendo as detecções

4.1. Processo simplificado

Método mais rápido e direto para obter detecções, recomendado para projetos pequenos. A função `template_matching()` processa automaticamente todos os arquivos e retorna as detecções. Usa 4 núcleos para processamento paralelo.

```

df_detections <- template_matching(
  soundscapes_path = "./soundscapes/",
  templates_path = "./templates/",
  ncores = 4
)

## Template metadata successfully extracted

## All files are compatible and included in the matching grid.

## Template matching finished. Detections have been returned to the R session

## Template matching finished

glimpse(df_detections)

```

```

## Rows: 792
## Columns: 21
## $ soundscape_path      <chr> "./soundscapes/W54393S25597_20201104_170000.wav"~
## $ soundscape_file       <chr> "W54393S25597_20201104_170000.wav", "W54393S2559~
## $ template_path          <chr> "./templates/Bcu_1_015.692-016.188s_03.151-09.23~
## $ template_file           <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~
## $ template_name            <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~
## $ template_min_freq        <dbl> 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, ~
## $ template_max_freq        <dbl> 9.232, 9.232, 9.232, 9.232, 9.232, 9.232, ~
## $ template_start            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ template_end              <dbl> 0.4963333, 0.4963333, 0.4963333, 0.4963333, 0.49~
## $ detection_start           <dbl> 0.5893122, 1.1401909, 2.6775271, 3.2796504, 3.90~
## $ detection_end             <dbl> 1.050513, 1.601392, 3.138728, 3.740851, 4.368597~
## $ detection_wl               <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## $ detection_ovlp              <dbl> 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, ~
## $ detection_sample_rate       <int> 24000, 24000, 24000, 24000, 24000, 24000, ~

```

```

## $ detection_buffer      <int> 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, ~
## $ detection_min_score   <dbl> NA, ~
## $ detection_min_quant    <dbl> NA, ~
## $ detection_top_n        <int> NA, ~
## $ peak_index              <dbl> 65, 108, 228, 275, 324, 377, 421, 472, 519, 603, ~
## $ peak_score               <dbl> 0.2584335, 0.2499991, 0.2765454, 0.2895299, 0.29~
## $ peak_quant                <dbl> 0.916, 0.874, 0.978, 0.994, 0.997, 0.984, ~

```

4.2. Processo detalhado: metadados das soundscapes

No processo detalhado, os metadados são processados separadamente, permitindo maior controle e filtragem antes do template matching. Começamos coletando informações das soundscapes (duração, taxa de amostragem, caminho, etc.).

```

df_soundscapes <- fetch_soundscape_metadata(
  soundscapes_path = "./soundscapes",
  recursive = TRUE, # Inclui subdiretórios
  ncores = 4 # Processamento paralelo
)
glimpse(df_soundscapes)

## Rows: 12
## Columns: 6
## $ soundscape_path      <chr> "./soundscapes/W54393S25597_20201104_170000.wav~
## $ soundscape_file       <chr> "W54393S25597_20201104_170000.wav", "W54431S256~
## $ soundscape_duration    <dbl> 59.99454, 59.99454, 59.99454, 59.99454, 59.9945~
## $ soundscape_sample_rate <int> 24000, 24000, 24000, 24000, 24000, 24000, 24000~
## $ soundscape_bitrate     <int> 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16~
## $ soundscape_layout       <chr> "mono", "mono", "mono", "mono", "mono", "mono", ~

```

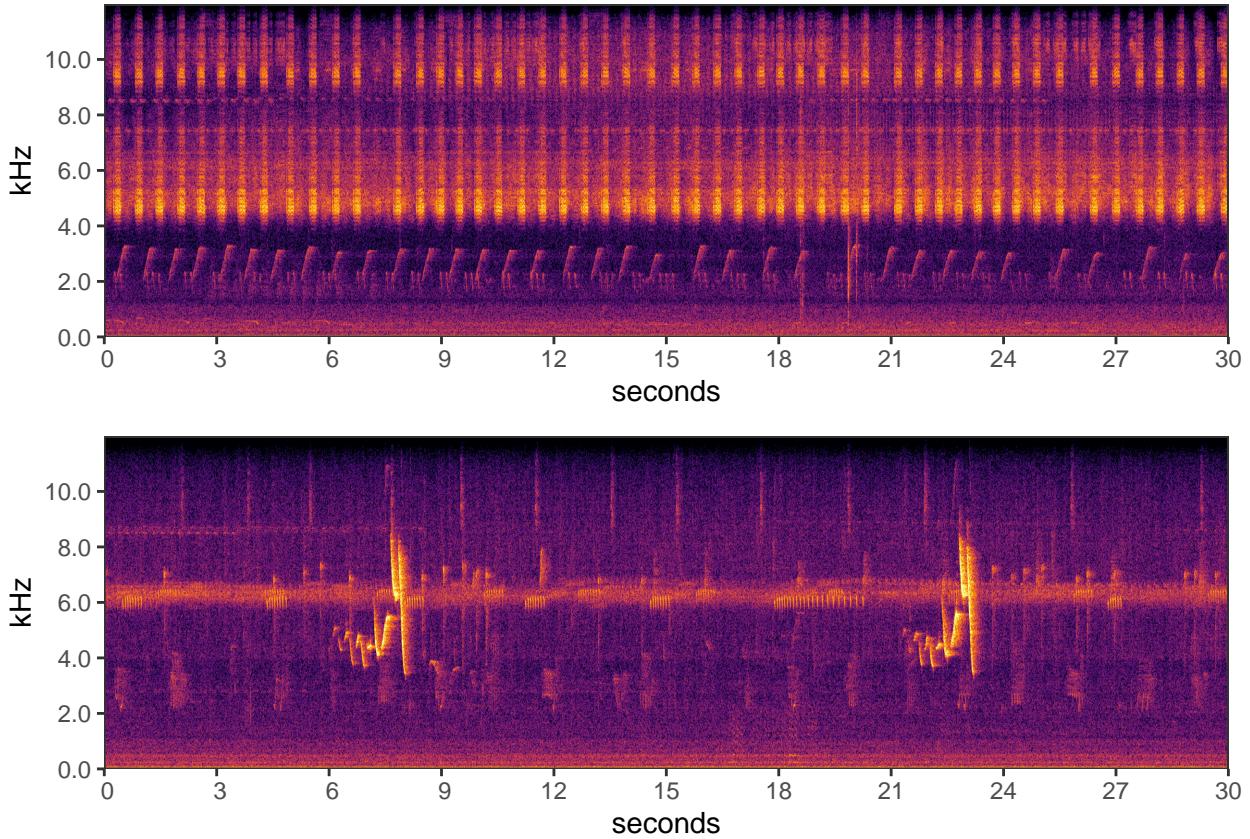
Visualização dos espectrogramas das duas primeiras soundscapes para verificar a importação correta.

```

## Loading required namespace: fftw

## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## i The deprecated feature was likely used in the monitoraSom package.
##   Please report the issue at
##   <https://github.com/ConservaSom/monitoraSom/issues>.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.

```



4.3. Processo detalhado: metadados dos templates

Coleta os metadados dos templates (duração, frequências, taxa de amostragem, etc.). Mesmo que os templates já tenham sido importados no passo 3.3, este passo garante que os metadados estejam completos e atualizados.

```
df_templates <- fetch_template_metadata(
  templates_path = "./templates/", recursive = TRUE
)

## Template metadata successfully extracted

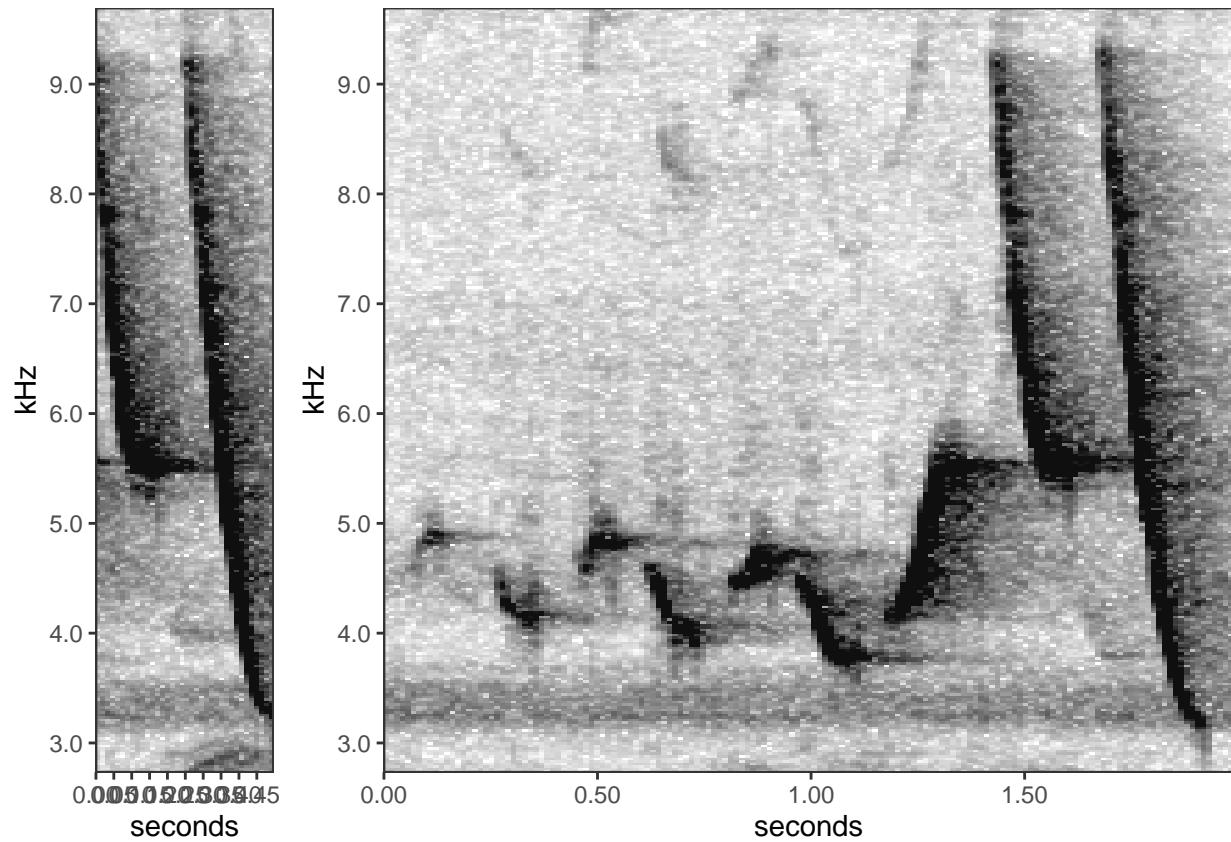
glimpse(df_templates)

## Rows: 2
## Columns: 11
## $ template_path      <chr> "./templates/Bcu_1_015.692-016.188s_03.151-09.232~"
## $ template_file       <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_7~"
## $ template_name        <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_7~"
## $ template_label       <chr> "Basileuterus culicivorus", "Basileuterus culiciv~"
## $ template_start        <dbl> 0, 0
## $ template_end          <dbl> 0.4963333, 1.9887500
## $ template_sample_rate <int> 24000, 24000
## $ template_min_freq    <dbl> 3.151, 2.733
```

```

## $ template_max_freq      <dbl> 9.232, 9.693
## $ template_wl            <dbl> 1024, 1024
## $ template_ovlp           <dbl> 70, 50

```



4.4. Processo detalhado: criar grade de busca

Cria uma grade combinando todas as combinações possíveis de soundscapes × templates. A função `fetch_match_grid()` verifica incompatibilidades (ex: taxas de amostragem diferentes) e permite filtrar dados antes do processamento, otimizando o template matching.

```

df_grid <- fetch_match_grid(
  soundscape_data = df_soundscapes, template_data = df_templates
)

## All files are compatible and included in the matching grid.

glimpse(df_grid)

## Rows: 24
## Columns: 17
## $ soundscape_path      <chr> "./soundscapes/W54393S25597_20201104_170000.wav~"
## $ soundscape_file       <chr> "W54393S25597_20201104_170000.wav", "W54393S255~"
## $ soundscape_duration   <dbl> 59.99454, 59.99454, 59.99454, 59.99454, 59.9945~

```

```

## $ soundscape_sample_rate <int> 24000, 24000, 24000, 24000, 24000, 24000, 24000~  

## $ soundscape_bitrate      <int> 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, ~  

## $ soundscape_layout        <chr> "mono", "mono", "mono", "mono", "mono", "mono", ~  

## $ template_path            <chr> "./templates/Bcu_1_015.692-016.188s_03.151-09.2~  

## $ template_file             <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl~  

## $ template_name             <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl~  

## $ template_label            <chr> "Basileuterus culicivorus", "Basileuterus culic~  

## $ template_start            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~  

## $ template_end              <dbl> 0.4963333, 1.9887500, 0.4963333, 1.9887500, 0.4~  

## $ template_sample_rate       <int> 24000, 24000, 24000, 24000, 24000, 24000, 24000~  

## $ template_min_freq          <dbl> 3.151, 2.733, 3.151, 2.733, 3.151, 2.733, 3.151~  

## $ template_max_freq          <dbl> 9.232, 9.693, 9.232, 9.693, 9.232, 9.693, 9.232~  

## $ template_wl                <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~  

## $ template_ovlp               <dbl> 70, 50, 70, 50, 70, 50, 70, 50, 70, 50, 70, 50, ~

```

4.5. Processo detalhado: executar template matching

Executa o template matching usando a grade criada no passo anterior. A função `run_matching()` oferece várias opções de filtragem e configuração, conforme detalhado abaixo.

```

run_matching(  

  df_grid = df_grid,  

  output_file = "./detections/df_deteecs.csv",  

  autosave_action = "replace", # Substitui arquivo existente  

  buffer_size = "template", # Evita detecções sobrepostas  

  ncores = 4  

)
df_deteecs <- read.csv(file = "./detections/df_deteecs.csv")
glimpse(df_deteecs)

```

Opções de autosave_action: - "replace": sobrescreve o arquivo existente - "append": adiciona resultados ao arquivo existente (cuidado com duplicatas)

Opções de buffer_size: - "template": buffer com duração igual ao template (evita sobreposições) - Valor numérico: quantidade de frames para o buffer - 0: desliga o buffer (não recomendado para validação manual)

Outras opções de filtragem: - `min_score`: score mínimo absoluto para detecção. **Cuidado:** pode eliminar templates/soundscapes sem detecções acima do limiar, impedindo cálculo correto de FN na validação (ver passo 5.1) - `min_quant`: quantil mínimo relativo (sempre retorna pelo menos uma detecção por busca) - `top_n`: quantidade fixa de detecções com maiores scores a serem retidas

4.6. Caso especial: obter scores brutos

Retorna os scores brutos (antes da filtragem) em vez das detecções processadas. Útil para análise detalhada e ajuste de limiares. O arquivo de saída será em formato RDS (não CSV).

```

run_matching(  

  df_grid = df_grid,  

  output = "scores",  

  output_file = "./detections/df_scores.rds",  

  ncores = 4  

)

```

```

## Template matching finished. Raw scores have been saved to ./detections/df_scores.rds

glimpse(df_scores)

## Rows: 72
## Columns: 20
## $ soundscape_path      <chr> "./soundscapes//W54393S25597_20201104_170000.wa~
## $ soundscape_file       <chr> "W54393S25597_20201104_170000.wav", "W54393S255~
## $ soundscape_duration   <dbl> 59.99454, 59.99454, 59.99454, 59.99454, 59.9945~
## $ soundscape_sample_rate <int> 24000, 24000, 24000, 24000, 24000, 24000, 24000~
## $ soundscape_bitrate    <int> 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, ~
## $ soundscape_layout     <chr> "mono", "mono", "mono", "mono", "mono", "mono", ~
## $ template_path          <chr> "./templates//Bcu_1_004.236-004.742s_03.151-09.~
## $ template_file           <chr> "Bcu_1_004.236-004.742s_03.151-09.201kHz_1024wl~
## $ template_name            <chr> "Basileuterus culicivorus", "Basileuterus culic~
## $ template_label           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ template_start            <dbl> 0.5053750, 0.4963333, 0.5098750, 0.3642500, 0.4~
## $ template_end              <int> 24000, 24000, 24000, 24000, 24000, 24000, 24000~
## $ template_sample_rate      <dbl> 3.151, 3.151, 3.057, 2.900, 2.975, 2.878, 3.151~
## $ template_min_freq         <dbl> 9.201, 9.232, 9.232, 9.764, 9.554, 9.636, 9.201~
## $ template_max_freq         <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## $ template_wl                <dbl> 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, ~
## $ template_ovlp               <int> 37, 36, 37, 26, 28, 28, 37, 36, 37, 26, 28, ~
## $ score_sliding_window        <chr> "cor", "cor", "cor", "cor", "cor", "cor", "cor"~
## $ score_method                 <list> [<data.frame[4684 x 2]>], [<data.frame[4684 x ~
## $ score_vec
```

Importa os scores brutos de volta ao ambiente. O objeto mantém a estrutura da grade de busca, mas a coluna `score_vec` contém uma lista com data frames de scores por frame (tempo e score) para cada combinação template-soundscape.

```

df_scores <- readRDS(file = "./detections/df_scores.rds")
glimpse(df_scores)
```

Inspeciona a estrutura dos scores brutos: cada elemento de `score_vec` é um data frame com `time_vec` (tempo em segundos) e `score_vec` (score do template matching para cada frame do espectrograma).

```
glimpse(df_scores$score_vec[[20]])
```

```

## Rows: 4,684
## Columns: 2
## $ time_vec  <dbl> 0.00000000, 0.01281113, 0.02562227, 0.03843340, 0.05124454, ~
## $ score_vec <dbl> -0.1029414, -0.1029414, -0.1029414, -0.1029414, ~
```

4.7. Visualizar os scores brutos

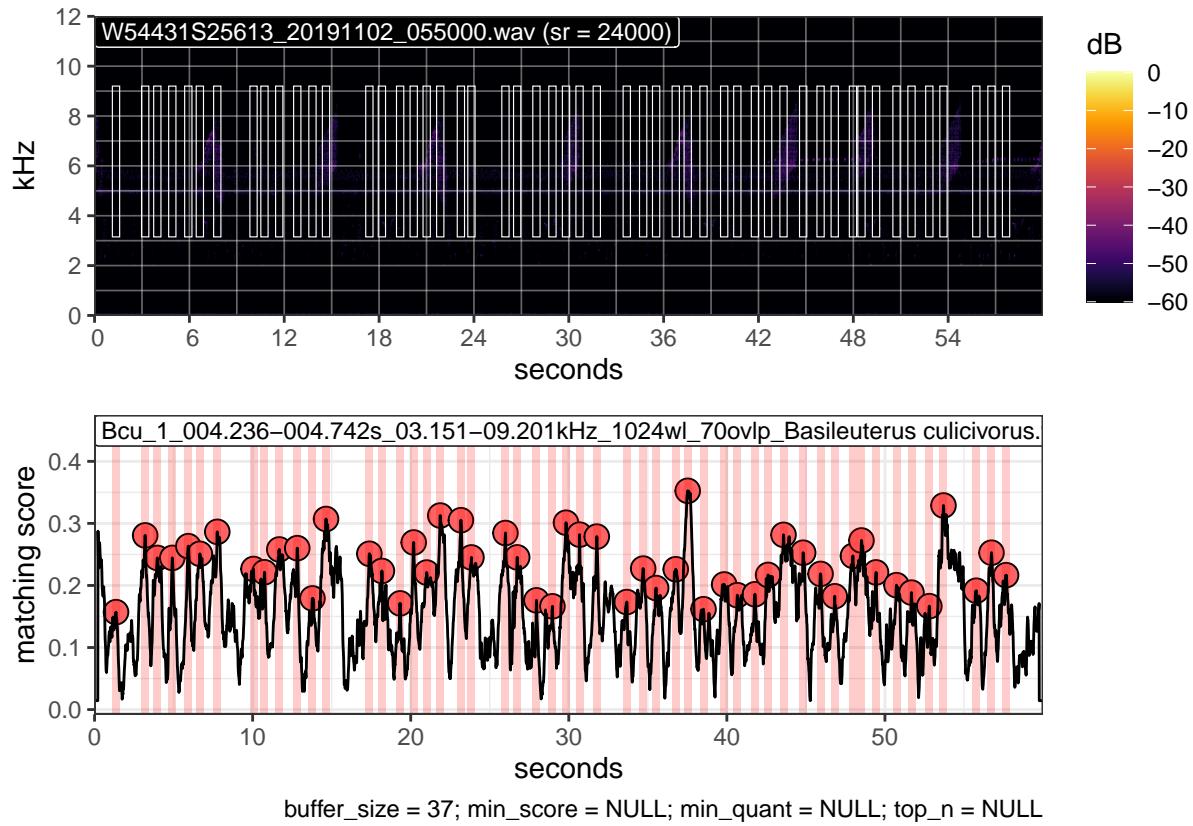
Visualiza os scores brutos ao longo do tempo para entender como as detecções foram geradas. A primeira visualização usa parâmetros padrão.

```

plot_scores(df_scores_i = df_scores[7, ])

## Coordinate system already present.
## i Adding new coordinate system, which will replace the existing one.

```

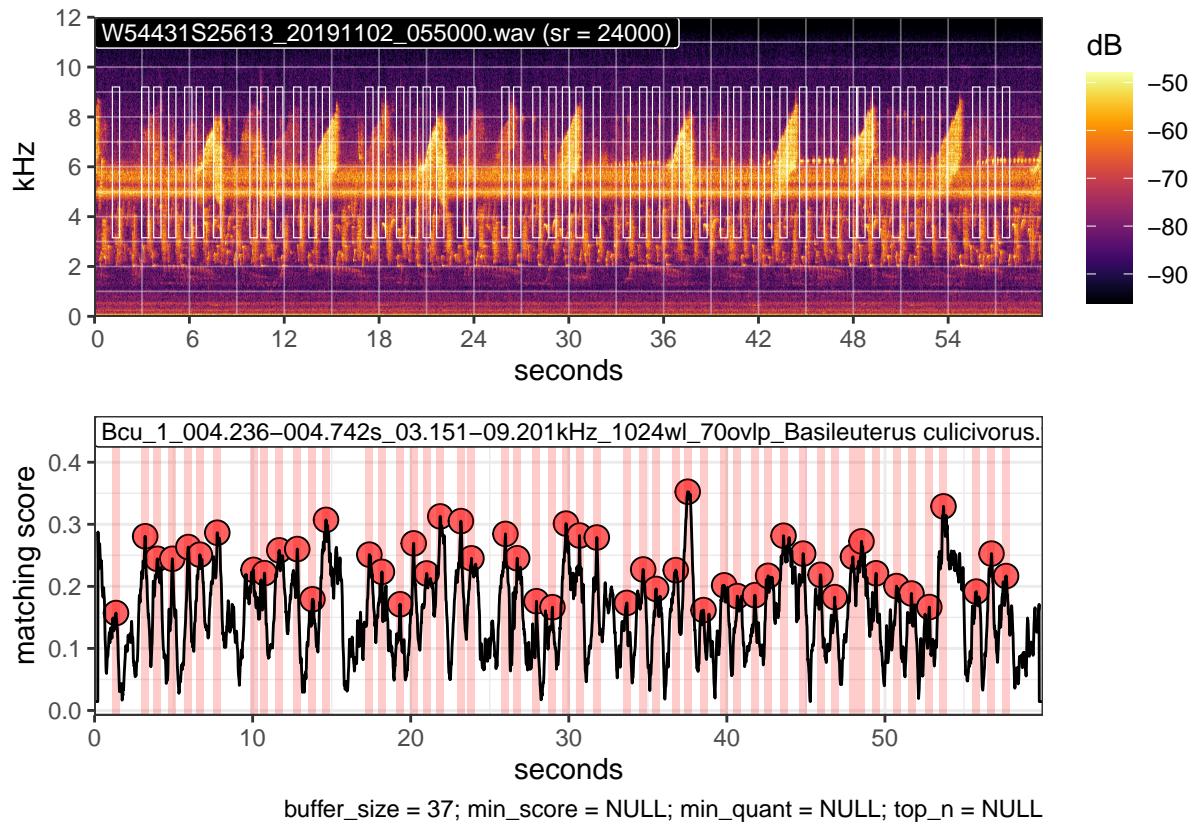


Visualização com parâmetros ajustados para melhor resolução e contraste.

```

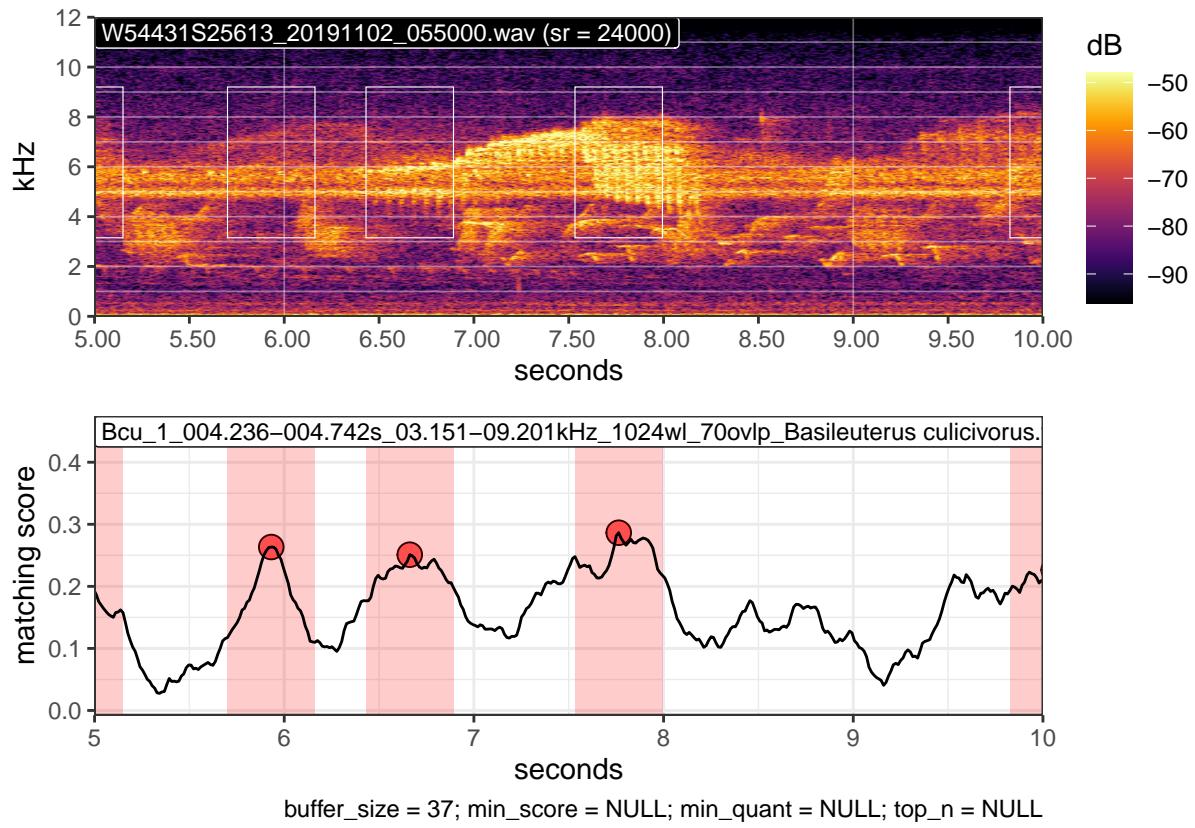
plot_scores(
  df_scores_i = df_scores[7, ], ovlp = 90, wl = 1024,
  dyn_range = c(-96, -48), color_scale = "inferno"
)

```



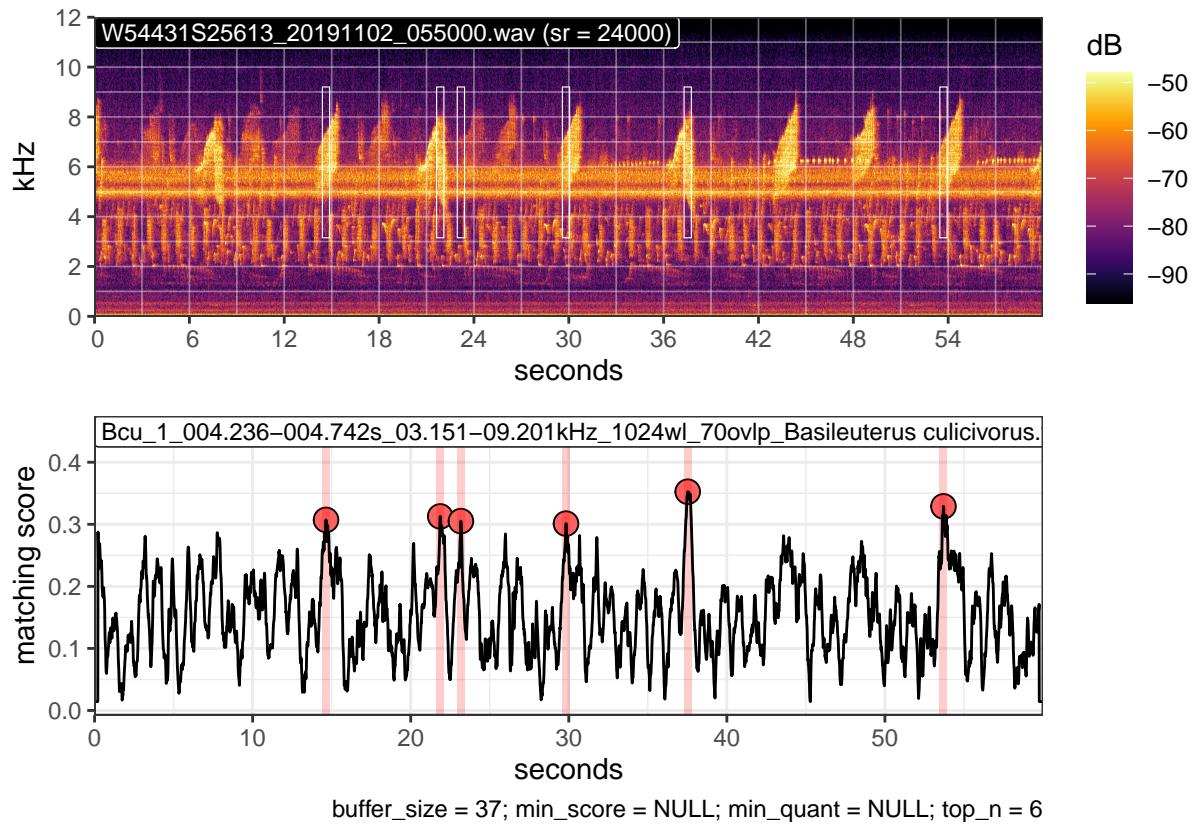
Zoom na região de tempo 5-10 segundos para visualizar um pico de detecção em detalhe.

```
plot_scores(
  df_scores_i = df_scores[7, ], ovlp = 70, wl = 1024,
  dyn_range = c(-96, -48), color_scale = "inferno", zoom_time = c(5, 10)
)
```



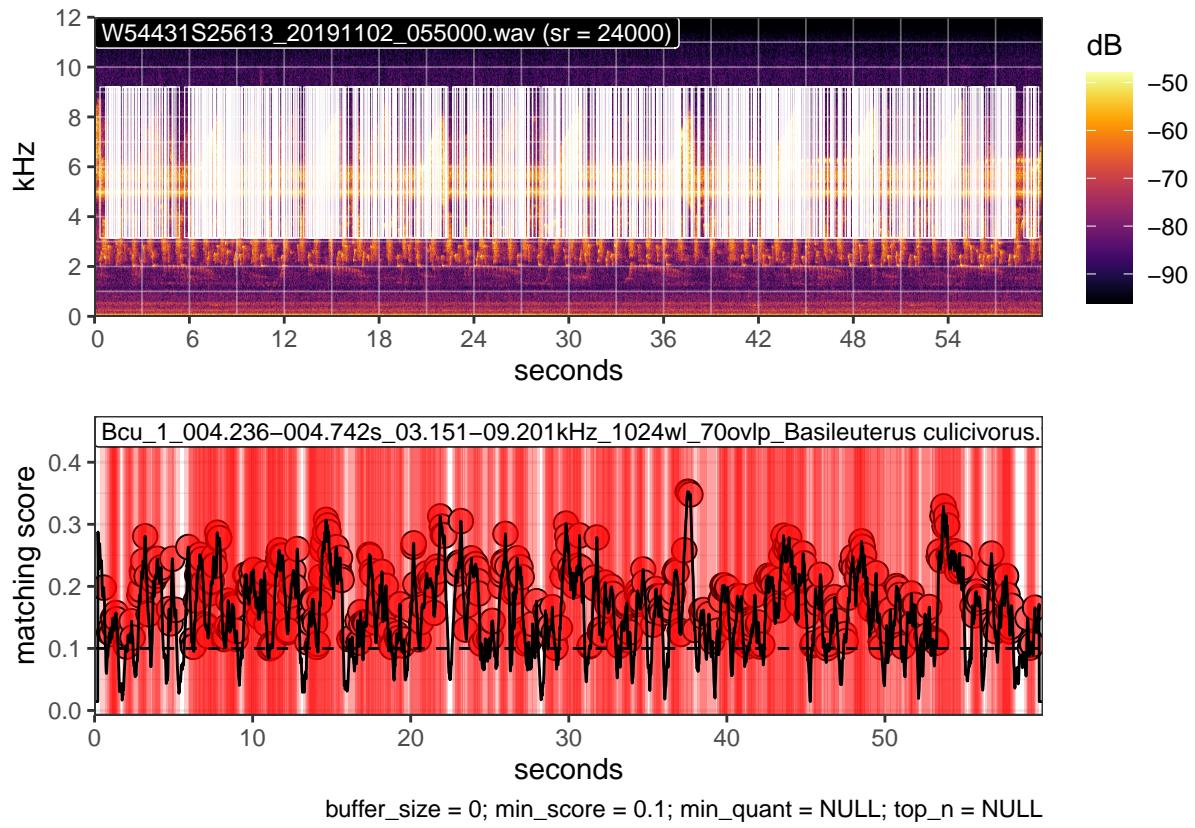
Aplica filtros diretamente na visualização: buffer ativo e retenção das 6 detecções com maiores scores (`top_n = 6`). Útil para testar diferentes configurações antes de processar todas as detecções.

```
plot_scores(
  df_scores_i = df_scores[7, ], ovlp = 70, wl = 1024,
  dyn_range = c(-96, -48), color_scale = "inferno",
  buffer_size = "template", top_n = 6
)
```



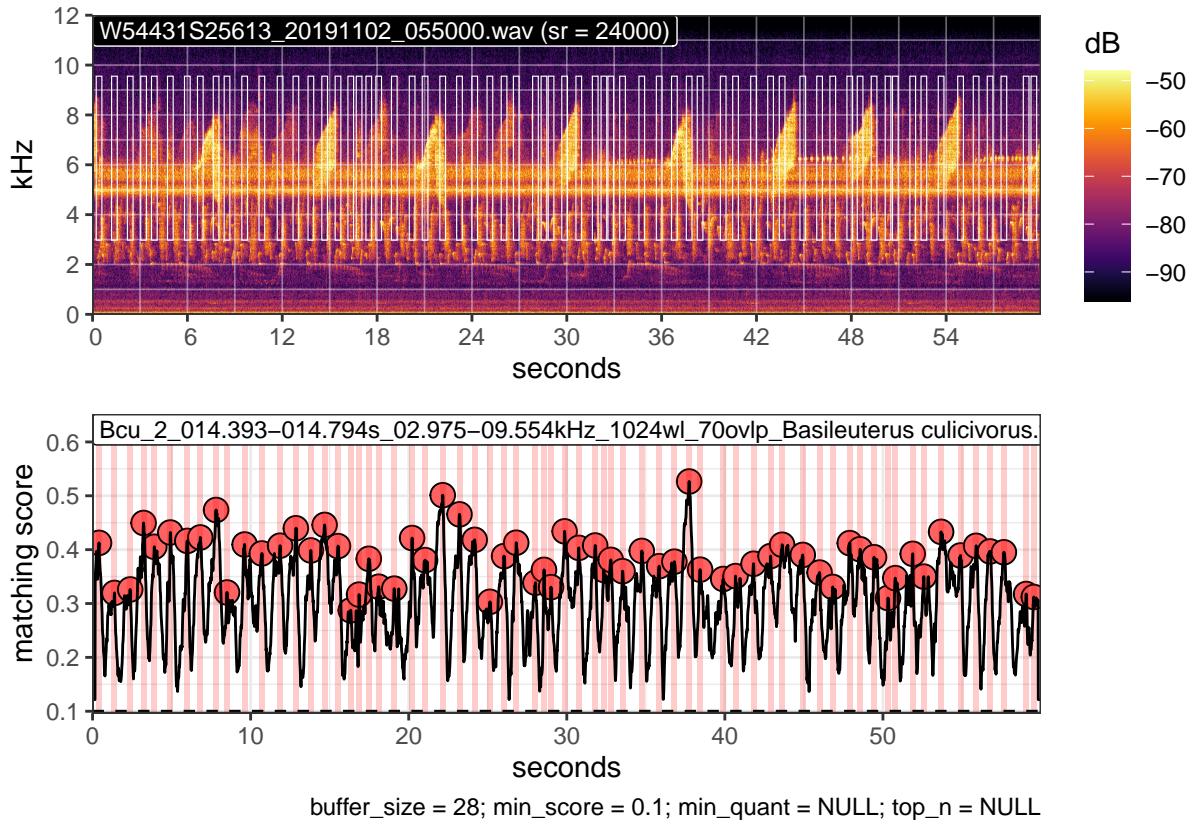
Testa filtro por score mínimo (`min_score = 0.1`) sem buffer. Neste exemplo, o limiar de 0.1 ainda retém algumas detecções falsas visíveis.

```
plot_scores(
  df_scores_i = df_scores[7, ], ovlp = 70, wl = 1024,
  dyn_range = c(-96, -48), color_scale = "inferno",
  buffer_size = 0, min_score = 0.1
)
```



Importante: Os parâmetros de filtragem que funcionam para uma soundscape podem não ser adequados para outras. A validação sistemática é essencial para definir limiares ideais.

```
plot_scores(
  df_scores_i = df_scores[11, ], ovlp = 70, wl = 1024,
  dyn_range = c(-96, -48), color_scale = "inferno",
  buffer_size = "template", min_score = 0.1
)
```



A avaliação visual é útil para ajuste inicial, mas a validação sistemática é necessária para definir limiares ideais. Extrai todas as detecções dos scores brutos sem filtros para validação completa.

```

df_detections_nofilt <- fetch_score_peaks(
  df_scores = df_scores,
  buffer_size = 0,
  output_file = "./detections/df_detections_nofilt.csv"
)

## Detections extracted from scores

## Detections have been exported to ./detections/df_detections_nofilt.csv

glimpse(df_detections_nofilt)

## Rows: 41,732
## Columns: 21
## $ soundscape_path      <chr> "./soundscapes//W54393S25597_20201104_170000.wav~"
## $ soundscape_file       <chr> "W54393S25597_20201104_170000.wav", "W54393S2559~"
## $ template_path         <chr> "./templates//Bcu_1_004.236-004.742s_03.151-09.2~"
## $ template_file          <chr> "Bcu_1_004.236-004.742s_03.151-09.201kHz_1024wl_~"
## $ template_name          <chr> "Bcu_1_004.236-004.742s_03.151-09.201kHz_1024wl_~"
## $ template_min_freq      <dbl> 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, ~
## $ template_max_freq      <dbl> 9.201, 9.201, 9.201, 9.201, 9.201, 9.201, 9.201, ~
## $ template_start          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~

```

```

## $ template_end          <dbl> 0.505375, 0.505375, 0.505375, 0.505375, 0.505375~
## $ detection_start        <dbl> 0.2562227, 0.3330895, 0.4740120, 0.5124454, 0.57~
## $ detection_end          <dbl> 0.7174235, 0.7942903, 0.9352128, 0.9736462, 1.03~
## $ detection_wl           <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## $ detection_ovlp          <dbl> 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, ~
## $ detection_sample_rate   <int> 24000, 24000, 24000, 24000, 24000, 24000, ~
## $ detection_buffer         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ detection_min_score      <dbl> NA, ~
## $ detection_min_quant      <dbl> NA, ~
## $ detection_top_n          <int> NA, ~
## $ peak_index                <dbl> 39, 45, 56, 59, 64, 85, 88, 100, 108, 131, 139, ~
## $ peak_score                 <dbl> 0.1690317, 0.1779760, 0.2104550, 0.2101202, 0.22~
## $ peak_quant                  <dbl> 0.415, 0.486, 0.746, 0.742, 0.878, 0.519, 0.521, ~

```

4.8. Checkpoint 04 - Filtragem adicional de detecções

Você obteve as detecções, mas veja que ainda não sabe quais são verdadeiras ou falsas. Nesse checkpoint você poderá fazer filtragens adicionais às demonstradas no passo acima segundo os critérios do seu projeto. Nesse ponto será necessário determinar qual método de validação será usado.

5. Validação das detecções

5.1. Validação a priori

Método automático baseado na sobreposição temporal entre detecções e ROIs pré-marcadas. Classifica cada detecção como TP (verdadeiro positivo), FP (falso positivo) ou FN (falso negativo). **Recomendado sempre que houver ROIs disponíveis**, pois permite contagem correta de FN e é muito mais rápido que validação manual.

```

df_validated <- validate_by_overlap(
  df_deteecs = df_deteecs, df_rois = df_rois, validation_user = "User"
)

## All detected species have ROIs for validation

## Validation results have been returned to the R session

glimpse(df_validated)

## Rows: 847
## Columns: 45
## $ soundscape_path          <chr> "soundscapes//Bcu_1.wav", "soundscapes//Bcu_1.wa~
## $ soundscape_file           <chr> "Bcu_1.wav", "Bcu_1.wav", "Bcu_1.wav", "Bcu_1.wa~
## $ template_path              <chr> "./templates/Bcu_1_015.692-016.188s_03.151-09.23~
## $ template_file               <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~
## $ template_name                <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~
## $ template_min_freq            <dbl> 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, ~
## $ template_max_freq            <dbl> 9.232, 9.232, 9.232, 9.232, 9.232, 9.232, 9.232, ~
## $ template_start                  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ template_end                    <dbl> 0.4963333, 0.4963333, 0.4963333, 0.4963333, 0.49~
## $ detection_start                  <dbl> NA, ~

```

```

## $ detection_end <dbl> NA, ~
## $ detection_wl <dbl> NA, ~
## $ detection_ovlp <dbl> NA, ~
## $ detection_sample_rate <int> NA, ~
## $ detection_buffer <int> NA, ~
## $ detection_min_score <dbl> NA, ~
## $ detection_min_quant <dbl> NA, ~
## $ detection_top_n <int> NA, ~
## $ peak_index <dbl> NA, ~
## $ peak_score <dbl> NA, ~
## $ peak_quant <dbl> NA, ~
## $ species <chr> "Basileuterus culicivorus", "Basileuterus culici~
## $ detection_id <int> NA, ~
## $ roi_path <chr> "/home/grosa/R_repos/2025_SimposioFisicaEcologia~
## $ roi_file <chr> "Bcu_1_roi_User_20250226101835.csv", "Bcu_1_roi_~
## $ roi_user <chr> "User", "User", "User", "User", "User", "User", ~
## $ roi_input_timestamp <chr> "2025-02-26 10:19:57", "2025-02-26 10:20:12", "2~
## $ roi_label <chr> "Basileuterus culicivorus", "Basileuterus culici~
## $ roi_start <dbl> 4.236420, 15.691662, 27.506793, 3.794958, 15.259~
## $ roi_end <dbl> 4.741796, 16.188014, 28.016681, 4.189475, 15.694~
## $ roi_min_freq <dbl> 3.150896, 3.150896, 3.057099, 3.659274, 3.561569~
## $ roi_max_freq <dbl> 9.200792, 9.232058, 9.232058, 5.629660, 5.743649~
## $ roi_type <chr> "bird - song", "bird - song", "bird - song", "bi~
## $ roi_label_confidence <chr> "certain", "certain", "certain", "cer~
## $ roi_is_complete <chr> "complete", "complete", "complete", "complete", ~
## $ roi_comment <chr> "Substructure C", "Substructure C", "Substructur~
## $ roi_wl <int> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## $ roi_ovlp <int> 70, 70, 70, 70, 70, 70, 70, 50, 70, ~
## $ roi_sample_rate <int> 24000, 24000, 24000, 24000, 24000, 24000, ~
## $ roi_pitch_shift <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ roi_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1~
## $ validation_user <chr> "User", "User", "User", "User", "User", "User", ~
## $ validation_time <chr> "2025-11-05 19:48:42", "2025-11-05 19:48:42", "2~
## $ validation <chr> "FN", "FN", "FN", "FN", "FN", "FN", "FN", ~
## $ validation_note <chr> "no detections to intersect with", "no detection~

```

5.2. Inspecionar resultados da validação

O objeto validado contém todas as variáveis das detecções originais, mais informações sobre as ROIs correspondentes. A coluna `validation` classifica cada detecção/ROI como: - TP: verdadeiro positivo (detecção correta) - FP: falso positivo (detecção incorreta) - FN: falso negativo (ROI não detectada)

Contagem de validações por template:

```
table(df_validated$validation, df_validated$template_name)
```

```

##          Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_70ovlp_Basileuterus culicivorus.wav   22
##          FN                                         22
##          FP                                         591
##          TP                                         48
##          Bcu_1_026.071-028.060s_02.733-09.693kHz_1024wl_50ovlp_Basileuterus culicivorus.wav

```

## FN	33
## FP	116
## TP	37

Comparação do número de detecções entre métodos:

```
nrow(df_deteecs)
```

```
## [1] 792
```

```
nrow(df_deteecs_nofilt)
```

```
## [1] 41732
```

Comparação de desempenho da validação a priori em diferentes cenários: o método é eficiente mesmo com grandes volumes de dados. A validação automática é muito mais rápida que a manual e permite validar todas as detecções sem esforço.

```
# Tempo de processamento: detecções filtradas
system.time({
  validate_by_overlap(
    df_deteecs = df_deteecs, df_rois = df_rois, validation_user = "User"
  )
})
```

```
## All detected species have ROIs for validation
```

```
## Validation results have been returned to the R session
```

```
##       user   system elapsed
##      0.065   0.000   0.065
```

```
# Tempo de processamento: detecções não filtradas
system.time({
  validate_by_overlap(
    df_deteecs = df_deteecs_nofilt, df_rois = df_rois, validation_user = "User"
  )
})
```

```
## All detected species have ROIs for validation
```

```
## Validation results have been returned to the R session
```

```
##       user   system elapsed
##      0.385   0.000   0.386
```

5.3. Validação a posteriori (manual)

Quando não há ROIs pré-marcadas, a validação deve ser feita manualmente usando o app interativo. Primeiro, fazemos uma cópia do arquivo de detecções para preservar os dados originais.

```

file.copy(
  from = "./detections/df_deteecs.csv",
  to = "./detections/df_deteecs_aposteriori.csv",
  overwrite = FALSE
)

```

Abre o app de validação manual. No app, você visualiza cada detecção junto com o template correspondente e classifica como TP ou FP. O resultado será salvo no arquivo ./detections/df_deteecs_aposteriori.csv.

```

launch_validation_app(
  project_path = ".",
  validation_user = "User",
  templates_path = "./templates/",
  soundscapes_path = "./soundscapes/",
  input_path = "./detections/df_deteecs_aposteriori.csv",
  output_path = "./detections/df_deteecs_aposteriori.csv",
  dyn_range_templ = c(-78, -30),
  dyn_range_detec = c(-78, -30),
  wl = 1024,
  ovlp = 70,
  time_guide_interval = 0,
  freq_guide_interval = 0,
  overwrite = TRUE
)

```

5.4. Importar resultados da validação a posteriori

Importa o arquivo de detecções validadas manualmente de volta ao ambiente de trabalho.

```

df_deteecs_aposteriori <- read.csv(
  file = "./detections/df_deteecs_aposteriori.csv"
)
glimpse(df_deteecs_aposteriori)

table(
  df_deteecs_aposteriori$validation,
  df_deteecs_aposteriori$template_name
)

```

5.5. Comparar validações a priori e a posteriori

Compara os resultados dos dois métodos de validação. Note que a validação a posteriori não permite contagem de FN, pois não há ROIs pré-marcadas como referência.

```

df_rois %>%
  filter(!grepl("Bcu", soundscape_file)) %>%
  nrow()

nrow(df_deteecs_aposteriori)

table(df_deteecs_aposteriori$validation, df_deteecs_aposteriori$template_name)

```

6. Diagnósticos de performance e resultados finais

6.1. Diagnósticos de performance

Avaliação de métricas de performance é essencial para otimizar o template matching:

- **Precision (precisão):** proporção de detecções que são verdadeiras
- **Recall (sensibilidade):** proporção de ocorrências do sinal que foram detectadas

Existe uma relação de trade-off: aumentar o limiar de score aumenta a precisão (menos FP) mas diminui o recall (mais FN), e vice-versa. A escolha do limiar ideal depende do objetivo: minimizar falsos positivos, maximizar detecções ou buscar um equilíbrio.

Calcula diagnósticos de performance para cada template, incluindo métricas em função de diferentes limiares de score.

```
ls_val_apriori <- diagnostic_validations(
  df_validated = df_validated, pos_prob = 0.90, val_a_priori = TRUE
)
```

```
## Validation diagnostics completed successfully.
```

```
# Extraí resultados para cada template
res_template1 <- ls_val_apriori[[1]]
res_template2 <- ls_val_apriori[[2]]
```

Inspeciona os diagnósticos: cada data frame contém métricas de performance (precision, recall, etc.) calculadas para diferentes limiares de score.

```
res_template1$diagnostics %>% glimpse()
```

```
## #> #> Rows: 639
## #> Columns: 12
## #> #> $ template_name <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_70ovlp_B~"
## #> #> $ peak_score <dbl> 0.6027716, 0.4985262, 0.4962222, 0.4902171, 0.4830211, 0~
## #> #> $ tp <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1~
## #> #> $ fp <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## #> #> $ tn <int> 591, 591, 591, 591, 591, 591, 591, 591, 591, 591, 591, 591, 5~
## #> #> $ fn <int> 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, ~
## #> #> $ precision <dbl> 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1~
## #> #> $ recall <dbl> 0.01428571, 0.02857143, 0.04285714, 0.05714286, 0.071428~
## #> #> $ sensitivity <dbl> 0.01428571, 0.02857143, 0.04285714, 0.05714286, 0.071428~
## #> #> $ specificity <dbl> 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1.0000~
## #> #> $ F1_score <dbl> 0.02816901, 0.05555556, 0.08219178, 0.10810811, 0.133333~
## #> #> $ selected <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, ~
```

```
res_template2$diagnostics %>% glimpse()
```

```
## #> #> Rows: 153
## #> Columns: 12
## #> #> $ template_name <chr> "Bcu_1_026.071-028.060s_02.733-09.693kHz_1024wl_50ovlp_B~"
## #> #> $ peak_score <dbl> 0.5992908, 0.5977625, 0.5824681, 0.5347159, 0.5290634, 0~
## #> #> $ tp <dbl> 1, 2, 3, 4, 4, 5, 6, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
## #> #> $ fp <dbl> 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ~
## #> #> $ tn <int> 116, 116, 116, 116, 115, 115, 115, 114, 114, 114, 114, 114, 1~
## #> #> $ fn <int> 69, 68, 67, 66, 65, 64, 64, 63, 62, 61, 60, 59, 58, ~
## #> #> $ precision <dbl> 1.0000000, 1.0000000, 1.0000000, 1.0000000, 0.8000000, 0~
## #> #> $ recall <dbl> 0.01428571, 0.02857143, 0.04285714, 0.05714286, 0.057142~
```

```

## $ sensitivity    <dbl> 0.01428571, 0.02857143, 0.04285714, 0.05714286, 0.057142~  

## $ specificity   <dbl> 1.0000000, 1.0000000, 1.0000000, 1.0000000, 0.9913793, 0~  

## $ F1_score       <dbl> 0.02816901, 0.05555556, 0.08219178, 0.10810811, 0.106666~  

## $ selected       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, ~

```

Visualiza os gráficos de performance para cada template:

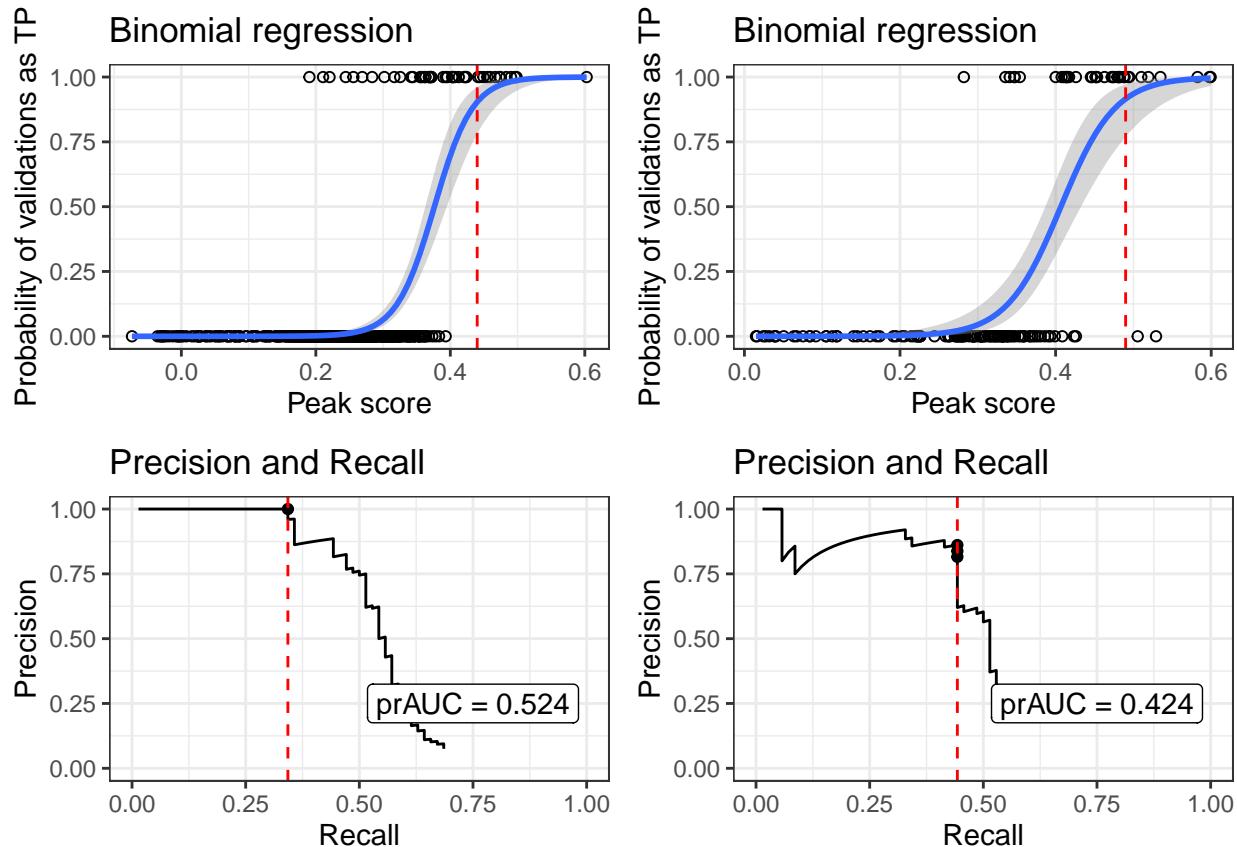
- **Coluna 1:** template 1 (subestrutura do canto)
- **Coluna 2:** template 2 (canto completo)
- **Linha 1:** modelos binomiais (probabilidade posterior de 95% de detecção verdadeira)
- **Linha 2:** precisão e recall em função dos limiares de score

As linhas vermelhas indicam os limiares de score selecionados para cada template.

```

cowplot::plot_grid(  
  res_template1$mod_plot, res_template2$mod_plot,  
  res_template1$precrec_plot, res_template2$precrec_plot,  
  ncol = 2  
)  


```



6.2. Obter conjunto final de detecções

Extrai os limiares de score ótimos calculados nos diagnósticos e filtra as detecções para manter apenas aquelas acima desses limiares.

6.2.1. Extrair limiares e nomes dos templates Extrai os limiares de score ótimos e os nomes dos templates dos resultados dos diagnósticos.

```

# Limiares de score ótimos para cada template
template1_score <- ls_val_apriori[[1]]$score_cut
template2_score <- ls_val_apriori[[2]]$score_cut

# Nomes dos templates
template1_name <- ls_val_apriori[[1]]$diagnostics$template_name[1]
template2_name <- ls_val_apriori[[2]]$diagnostics$template_name[1]

```

6.2.2. Filtrar detecções do template 1 Filtra as detecções mantendo apenas aquelas do template 1 com score maior ou igual ao limiar ótimo.

```

df_deteecs_final_1 <- df_deteecs %>%
  filter(
    template_name == template1_name & peak_score >= template1_score
  ) %>%
  glimpse()

## #> Rows: 13
## #> Columns: 21
## #> $ soundscape_path      <chr> "./soundscapes/W54431S25613_20191104_154000.wav"~
## #> $ soundscape_file       <chr> "W54431S25613_20191104_154000.wav", "W54431S2561~"
## #> $ template_path          <chr> "./templates/Bcu_1_015.692-016.188s_03.151-09.23~"
## #> $ template_file           <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~"
## #> $ template_name            <chr> "Bcu_1_015.692-016.188s_03.151-09.232kHz_1024wl_~"
## #> $ template_min_freq        <dbl> 3.151, 3.151, 3.151, 3.151, 3.151, 3.151, ~
## #> $ template_max_freq        <dbl> 9.232, 9.232, 9.232, 9.232, 9.232, 9.232, ~
## #> $ template_start           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## #> $ template_end             <dbl> 0.4963333, 0.4963333, 0.4963333, 0.49~
## #> $ detection_start          <dbl> 7.597003, 22.739763, 33.680472, 46.465984, 10.03~
## #> $ detection_end            <dbl> 8.058203, 23.200964, 34.141673, 46.927185, 10.49~
## #> $ detection_wl              <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## #> $ detection_ovlp            <dbl> 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, ~
## #> $ detection_sample_rate     <int> 24000, 24000, 24000, 24000, 24000, 24000, ~
## #> $ detection_buffer           <int> 36, 36, 36, 36, 36, 36, 36, 36, 36, ~
## #> $ detection_min_score        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## #> $ detection_min_quant         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## #> $ detection_top_n             <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## #> $ peak_index                  <dbl> 612, 1794, 2648, 3646, 802, 2297, 1983, 2349, ~
## #> $ peak_score                   <dbl> 0.4830211, 0.4985262, 0.4731373, 0.4962222, 0.44~
## #> $ peak_quant                  <dbl> 0.999, 1.000, 0.999, 1.000, 1.000, 1.000, ~

```

Visualização das detecções do template 1.

6.2.3. Filtrar detecções do template 2 Filtra as detecções mantendo apenas aquelas do template 2 com score maior ou igual ao limiar ótimo.

```

df_deteecs_final_2 <- df_deteecs %>%
  filter(
    template_name == template2_name & peak_score >= template2_score
  ) %>%
  glimpse()

```

```

## Rows: 10
## Columns: 21
## $ soundscape_path      <chr> "./soundscapes/W54448S25622_20191101_073000.wav"~
## $ soundscape_file       <chr> "W54448S25622_20191101_073000.wav", "W54448S2562~
## $ template_path         <chr> "./templates/Bcu_1_026.071-028.060s_02.733-09.69~
## $ template_file          <chr> "Bcu_1_026.071-028.060s_02.733-09.693kHz_1024wl_~
## $ template_name          <chr> "Bcu_1_026.071-028.060s_02.733-09.693kHz_1024wl_~
## $ template_min_freq      <dbl> 2.733, 2.733, 2.733, 2.733, 2.733, 2.733, ~
## $ template_max_freq      <dbl> 9.693, 9.693, 9.693, 9.693, 9.693, 9.693, ~
## $ template_start          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0
## $ template_end            <dbl> 1.98875, 1.98875, 1.98875, 1.98875, 1.9~
## $ detection_start          <dbl> 8.006389, 19.407487, 28.801650, 40.437602, 43.34~
## $ detection_end            <dbl> 9.970623, 21.371721, 30.765884, 42.401836, 45.30~
## $ detection_wl              <dbl> 1024, 1024, 1024, 1024, 1024, 1024, 1024, ~
## $ detection_ovlp            <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50
## $ detection_sample_rate     <dbl> 24000, 24000, 24000, 24000, 24000, 24000, ~
## $ detection_buffer           <dbl> 92, 92, 92, 92, 92, 92, 92, 92, 92
## $ detection_min_score        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ detection_min_quant        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ detection_top_n             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ peak_index                  <dbl> 422, 956, 1396, 1941, 2077, 2339, 2521, 1979, 28~
## $ peak_score                   <dbl> 0.5977625, 0.5992908, 0.5824681, 0.5191764, 0.53~
## $ peak_quant                  <dbl> 1.000, 1.000, 0.998, 0.980, 0.989, 0.972, 0.987, ~

```

Visualização das detecções do template 2.

6.3. Converter detecções finais em tabelas de ROIs

Converte as detecções filtradas em formato de tabelas de ROIs, que podem ser exportadas e usadas em outros aplicativos ou análises.

```

dir.create("./final_deteecs", showWarnings = FALSE)
df_final <- rbind(df_deteecs_final_1, df_deteecs_final_2) %>% glimpse()
df_rois_final <- detecs_to_rois(
  df_deteecs = df_final, username = "User",
  output_path = "./final_deteecs/"
)

```

Considerações finais

Resumo do processo

Ao longo deste tutorial, realizamos o processamento completo de detecções acústicas utilizando dois templates distintos. O fluxo incluiu: (1) preparação e organização dos dados de entrada (templates e soundscapes), (2) execução do template matching usando diferentes abordagens (simplificada e detalhada), (3) validação das detecções através de métodos a priori ou a posteriori, (4) análise de performance e otimização de limiares, e (5) geração de resultados finais em formatos reutilizáveis.

A estrutura de diretórios e organização dos dados ao longo do processo permite que os resultados sejam revisitados de forma rápida e eficiente, bem como adaptados para diferentes contextos de uso. Os arquivos intermediários (metadados, scores brutos, detecções filtradas) podem ser utilizados para análises adicionais ou ajustes posteriores sem necessidade de reprocessar todo o pipeline.

Adaptação para diferentes projetos

Este fluxo de análise pode ser adaptado para diferentes projetos em termos de escala e complexidade:

- **Monitoramento de populações:** pode requerer o uso de múltiplos templates quando a espécie alvo apresentar repertórios vocais mais complexos, além da aplicação de filtros adicionais para reduzir o número de detecções falsas. A validação sistemática se torna ainda mais importante nesses casos.
- **Monitoramento de comunidades:** adiciona uma camada extra de complexidade devido à necessidade de detecção de múltiplas espécies simultaneamente. Nesse contexto, recomenda-se processar cada espécie separadamente ou criar pipelines paralelos para diferentes grupos taxonômicos.
- **Projetos em larga escala:** para grandes volumes de dados, considere usar o processo detalhado (seções 4.2-4.5) para maior controle sobre a filtragem e processamento, bem como usar o processamento em paralelo para maior eficiência.

Próximos passos e extensões

Algumas coisas a explorar quando começar a usar o monitoraSom para seu próprio projeto:

- **Ajuste fino de parâmetros:** experimente diferentes configurações de `wl`, `ovlp`, `dyn_range` e `buffer_size` para otimizar a detecção de acordo com as características acústicas específicas do seu projeto.
- **Integração com outros pacotes:** os dados exportados podem ser facilmente integrados com outros pacotes do R para análise de morfometria, modelagem ecológica, ecologia espacial, fenologia, etc.
- **Automação de workflows:** para projetos repetitivos, considere criar scripts automatizados que combinem múltiplas etapas do processo, reduzindo o tempo de processamento e aumentando a reproduzibilidade.

Boas práticas recomendadas

1. **Documentação:** mantenha scripts organizados e registros dos parâmetros utilizados em cada etapa, especialmente limiares de score e configurações de filtragem.
2. **Validação contínua:** mesmo com validação a priori, revise a performance dos templates periodicamente, especialmente após a entrada de novos conjuntos de dados para garantir que os resultados não se deteriorem como consequência de mudanças ambientais..
3. **Backup de dados:** preserve os arquivos de scores brutos e detecções não filtradas, pois permitem reavaliação sem reprocessamento completo.
4. **Controle de versão:** use sistemas de controle de versão (ex: Git) para rastrear mudanças em scripts e parâmetros ao longo do projeto.

Supporte e contribuições

Nós, os desenvolvedores do monitoraSom, estamos sempre abertos a sugestões e melhorias para o pacote. Se você tiver alguma ideia, encontrar um bug, ou quiser contribuir com o desenvolvimento, por favor, entre em contato conosco através do nosso canal de comunicação do GitHub: <https://github.com/ConservaSom/monitoraSom/issues>.

Para documentação adicional, exemplos avançados e atualizações do pacote, visite nosso repositório: <https://github.com/ConservaSom/monitoraSom>.

Para informações mais completas e embasamento teórico sobre o monitoraSom, visite nosso preprint: <https://www.biorxiv.org/content/10.1101/2025.06.23.661148v1>.

Boa sorte nas descobertas a seguir!