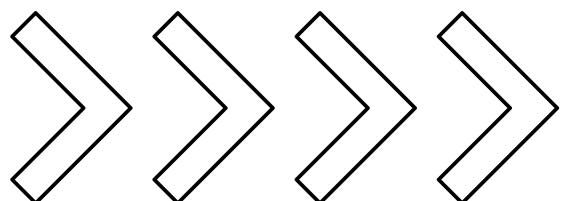


APRESENTAÇÃO

TRABALHO 1

DETECTOR DE ASSOvio DE GOLFINHO

REDES NEURAIS E DEEP LEARNING



Pós-Graduando: Efrain Marcelo
Docente: Prof. Dr. Adrião Duarte
Curso: PPGEEC

Índice

01 Introdução

02 Objetivo

03 Metodologia

04 Desenvolvimento

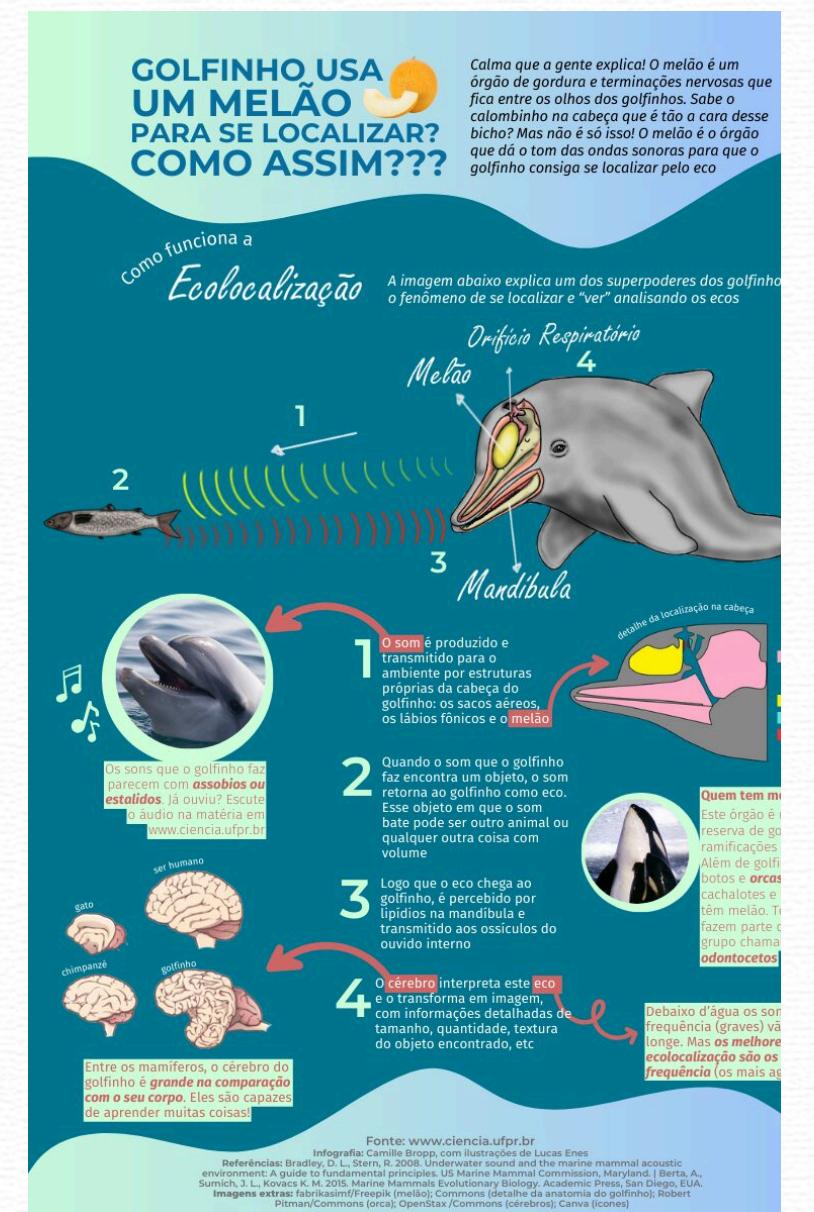
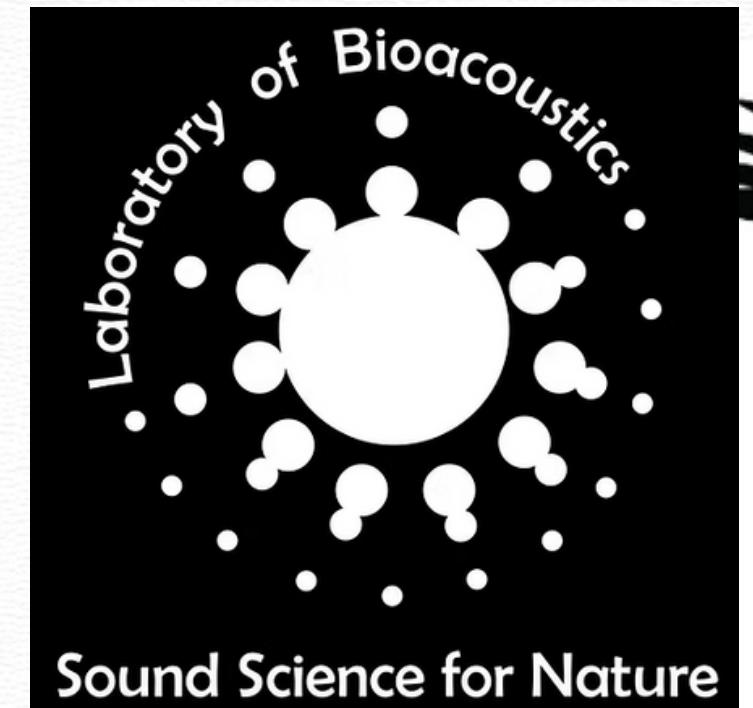
05 Solução Final

06 Conclusão

Introdução

Assovios de Golfinhos

- Demanda proveniente do Laboratório de Pesquisa em Bioacústica.
- Analistas utilizam o software Raven 1.6 para identificação de assovios de golfinhos.
- Processo atual é computacionalmente pesado e extremamente demorado.
- Dificuldade em escalar a análise para grandes volumes de dados.

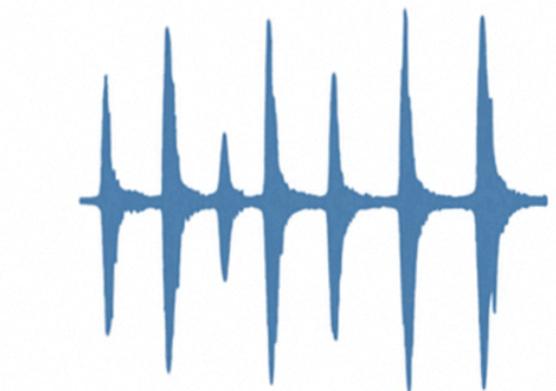
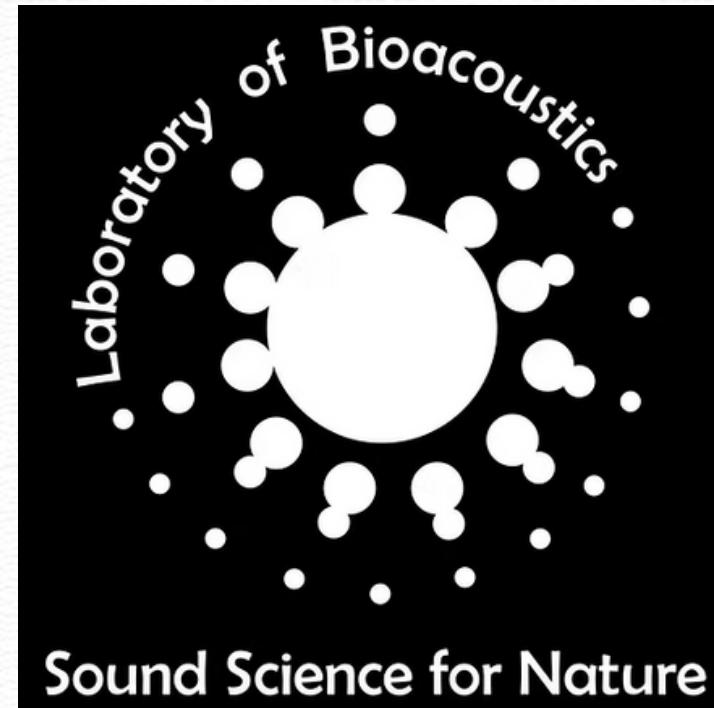


-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

Objetivo

Desenvolver um detector automático de assobios utilizando Python.

- Python oferece maior flexibilidade: aplicações embarcadas, servidores, notebooks pessoais, etc.
- Busca por agilidade, eficiência e acessibilidade no processamento dos dados acústicos.
- Uso de técnicas de Aprendizado de Máquina para detectar assobios e descobrir padrões de comportamento acústico.
- Projeto visa também escalar a metodologia para outras classes de eventos bioacústicos.
-



Whistle





Metodologia

01 Fonte de Dados

02 Análise Exploratória

03 Extração de Características (MFCC)

04 Criar Dataset

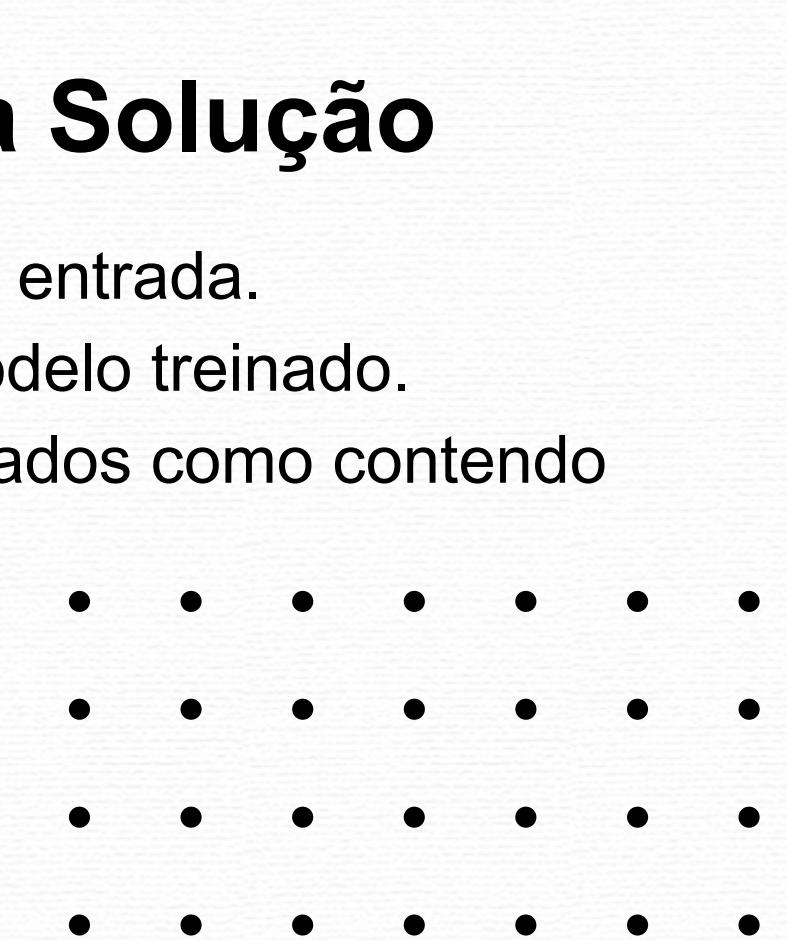
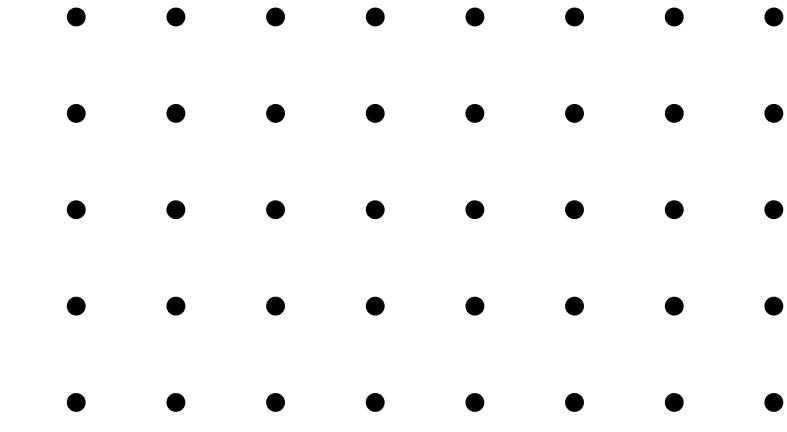
- Extração de MFCCs para janelas com eventos (assovios) e sem eventos.
- Criação de uma base balanceada com instâncias positivas e negativas.

05 Seleção do Modelo

- Avaliação de diversos algoritmos via Lazy Predict (acurácia × tempo de treinamento).
- Escolha do modelo mais leve e eficaz (LGBMClassifier).

06 Implantação da Solução

- Recebe um áudio bruto como entrada.
- Executa a predição com o modelo treinado.
- Retorna os instantes classificados como contendo assovios.



Desenvolvimento

Treinamento Inicial

- Modelo treinado com todas as features extraídas dos MFCCs.
- Execução de validação e avaliação dos resultados.

Features Importance

- Após o treinamento inicial, analisamos a importância relativa das variáveis para a decisão do modelo.
- Identificação das 6 features mais relevantes.

```
# Separar features (X) e target (y)
X = df_scaled.iloc[:, :-1] # Todas as colunas menos a última
y = df_scaled.iloc[:, -1] # Última coluna (Class)

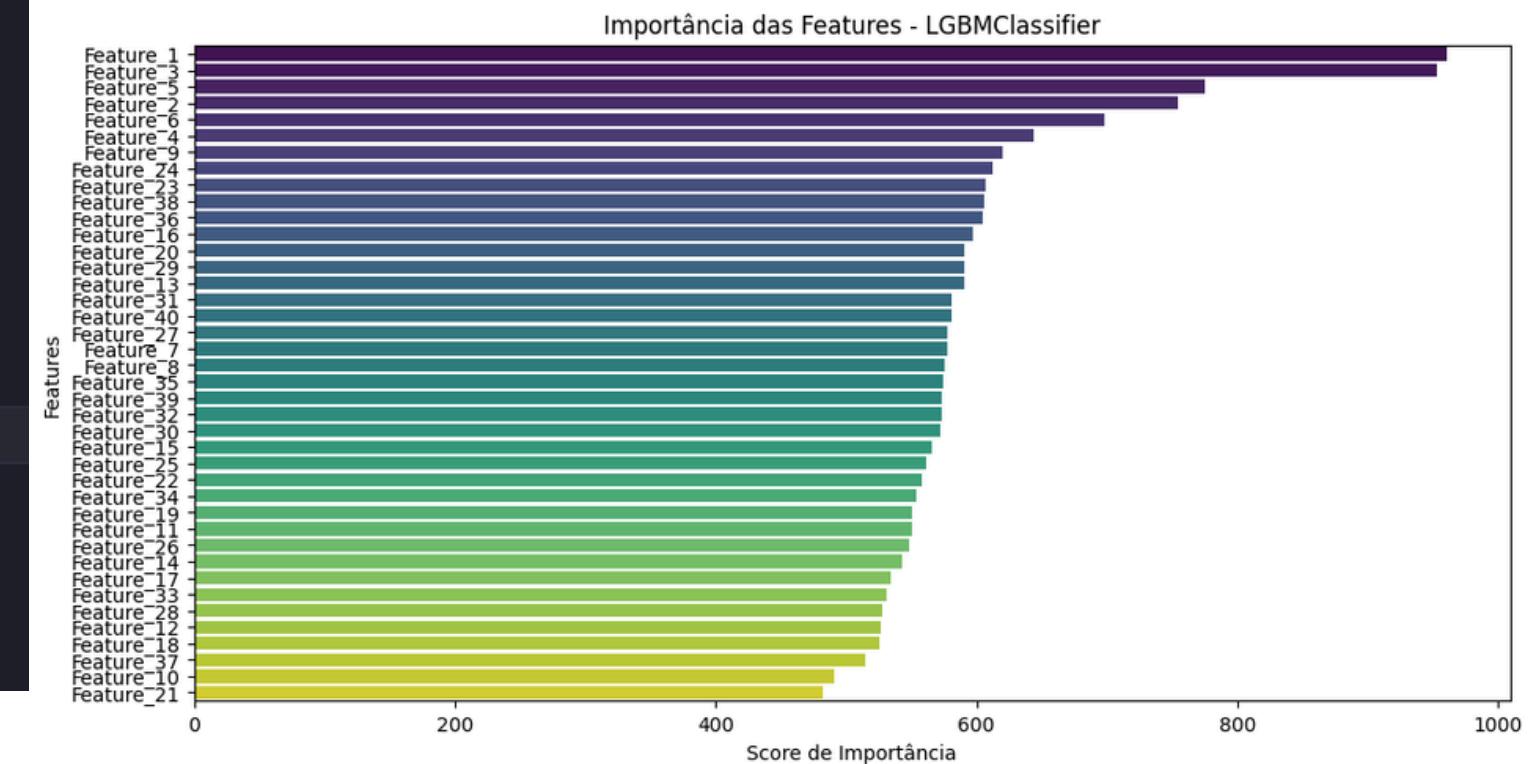
# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.7,
                                                    random_state=42)

[3]

model = lgb.LGBMClassifier(
    n_estimators=1000,
    max_depth=6,
    num_leaves=25,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=1.0,
    reg_lambda=1.0,
    random_state=42
)

[4]

model.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)]
)
```



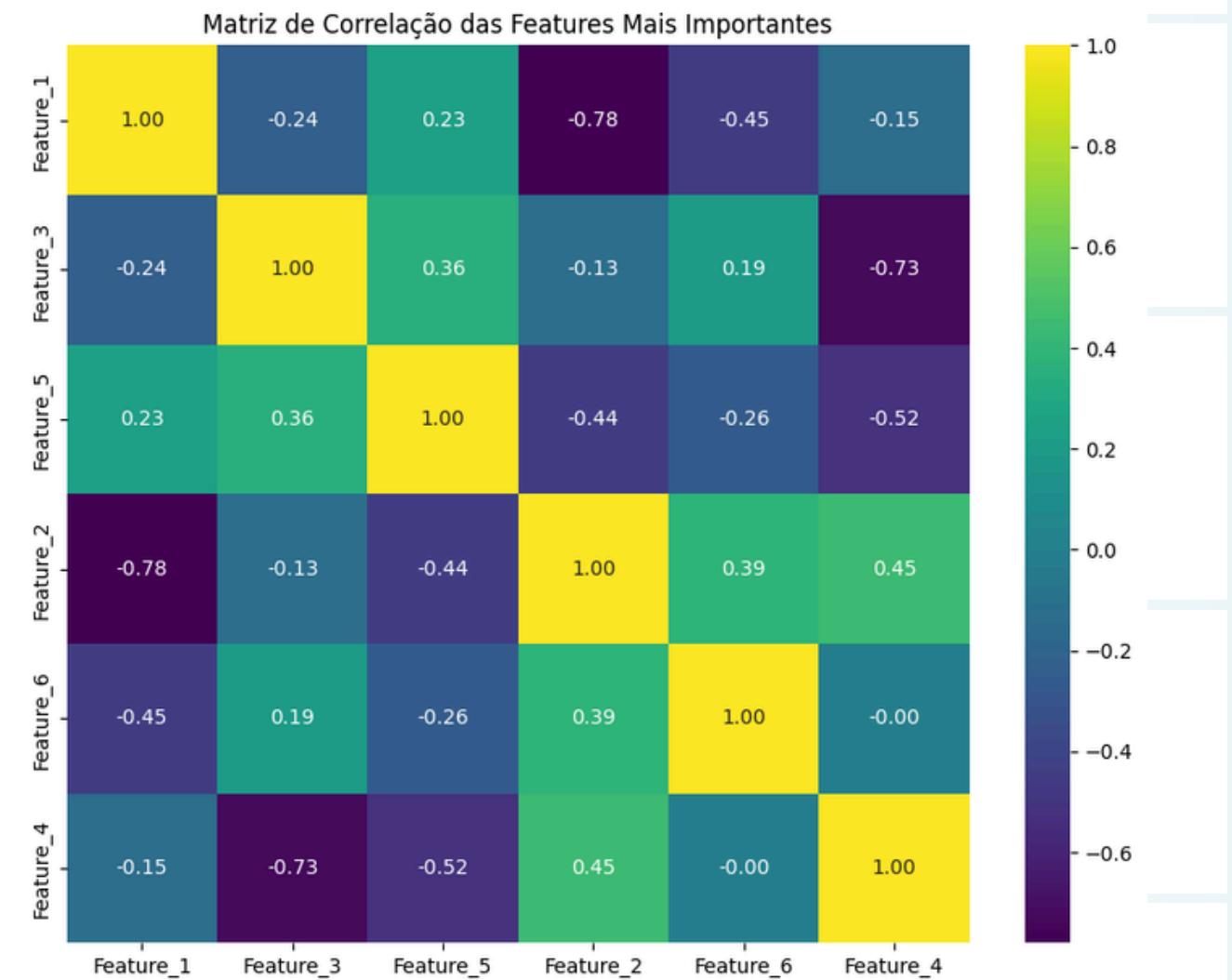
	Relatório de Classificação:				
	precision	recall	f1-score	support	
0	0.92	0.99	0.96	7162	
1	0.99	0.92	0.95	7241	
accuracy				0.96	14403
macro avg	0.96	0.96	0.96	14403	
weighted avg	0.96	0.96	0.96	14403	

```
LGBMClassifier
LGBMClassifier(colsample_bytree=0.8, max_depth=6, n_estimators=1000,
               num_leaves=25, random_state=42, reg_alpha=1.0, reg_lambda=1.0,
               subsample=0.8)
```

Desenvolvimento

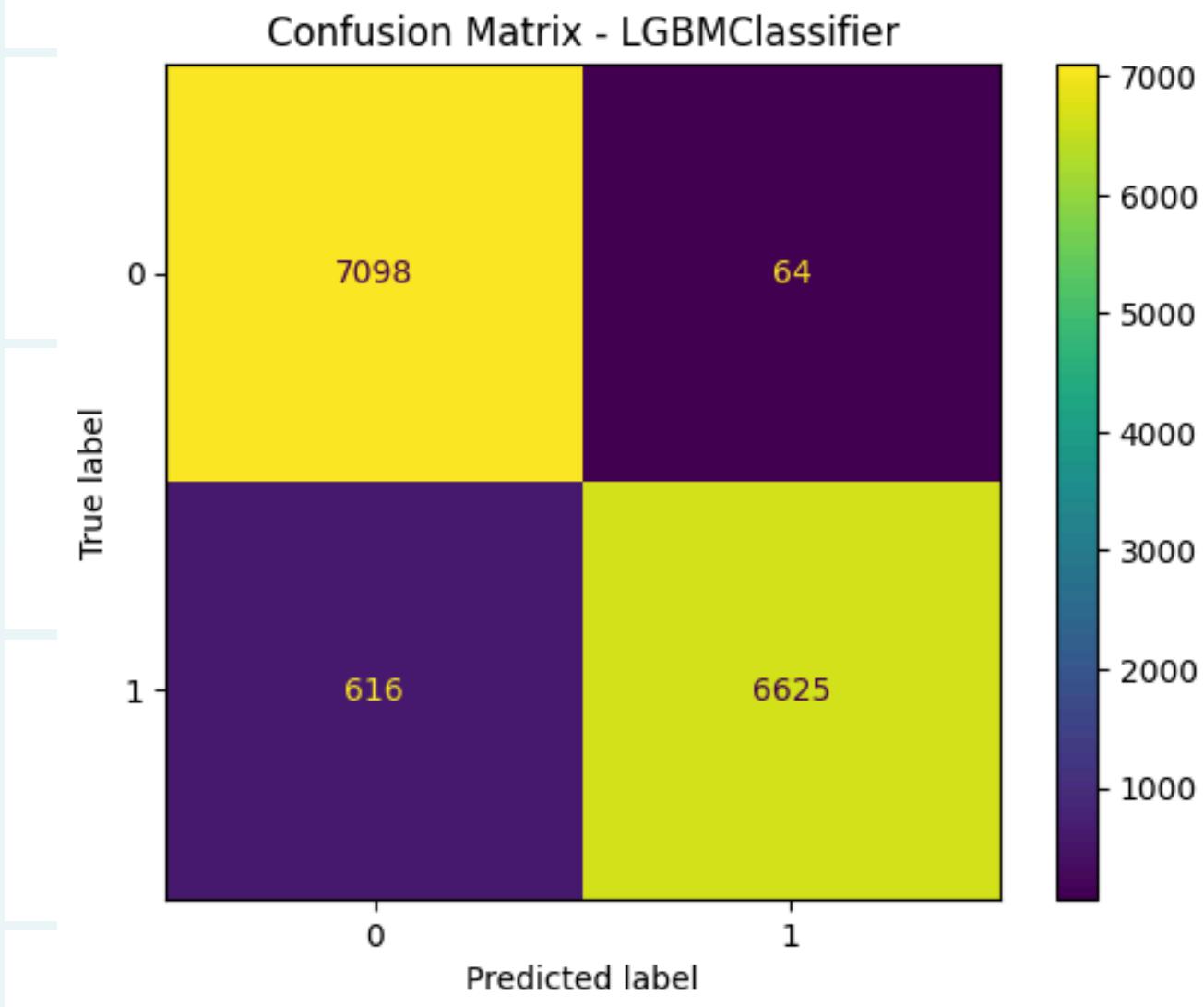
Modelo Otimizado (Versão Final)

- Re-treinamento do LGBMClassifier utilizando apenas as 6 variáveis principais.
- Menor tempo de inferência, mantendo excelente desempenho.

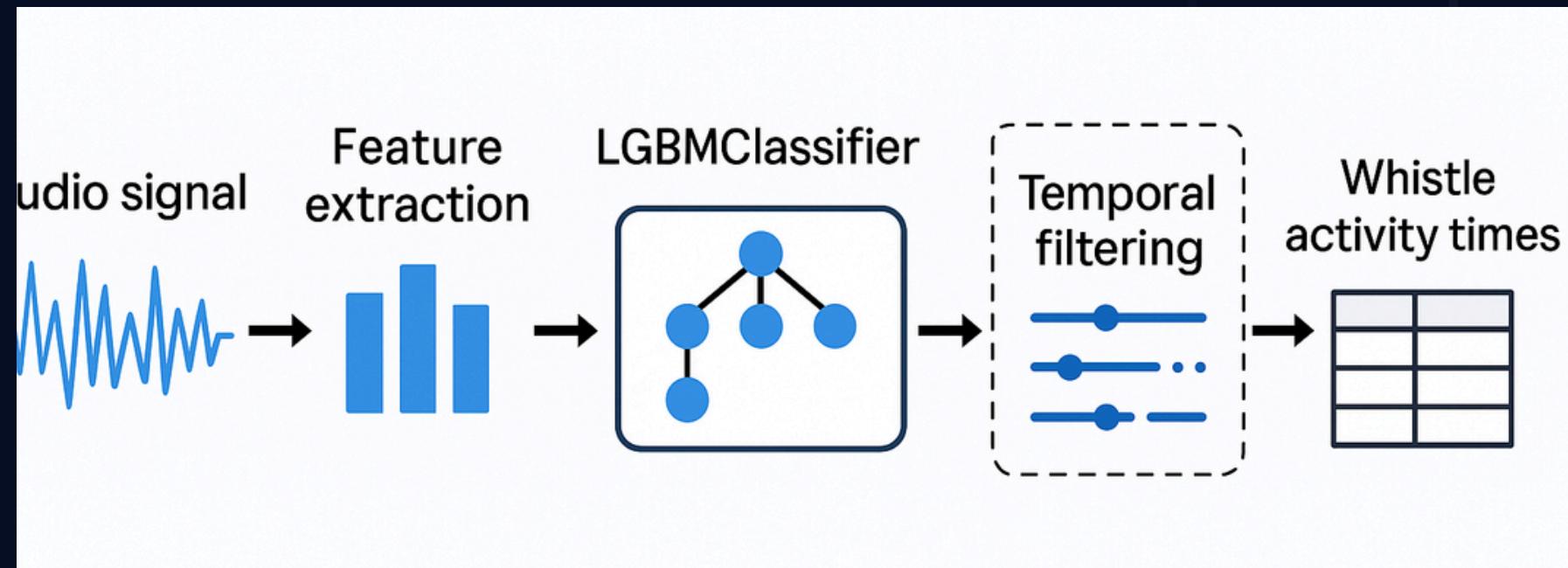


Relatório de Classificação:

	precision	recall	f1-score	support
0	0.92	0.99	0.95	7162
1	0.99	0.91	0.95	7241
accuracy			0.95	14403
macro avg	0.96	0.95	0.95	14403
weighted avg	0.96	0.95	0.95	14403



Solução Final



Aplicação de Filtro Temporal

- Agrupa detecções próximas dentro de uma janela de tempo (ex: 0.5s).
- Exige mínimo de 3 detecções consecutivas para considerar um evento real.
- Reduz falsos positivos e suaviza flutuações aleatórias da rede.
- Saída: tabela com os intervalos reais de assobios identificados no áudio.

```
def filter_consecutive_detections_with_group(df, min_detections=3, max_gap=0.5):    Refactor this function to reduce it
    """
    Filtra grupos de detecções que tenham pelo menos `min_detections` em um intervalo total de até `max_gap` segundos.
    Atribui identificadores únicos (group_id) aos grupos válidos.

    Parâmetros:
    - df (pd.DataFrame): DataFrame com colunas ['filename', 'time']
    - min_detections (int): Mínimo de detecções no grupo
    - max_gap (float): Janela máxima (em segundos) entre a primeira e última detecção do grupo

    Retorno:
    - pd.DataFrame com colunas ['filename', 'time', 'group_id']
    """
    # Implementation details...
```

Saida

Tabela de Detecções e Melhorias

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse laoreet dolor sapien, ac volutpat diam faucibus sed. Maecenas imperdiet viverra risus, pharetra mattis lectus vestibulum vel. Donec sollicitudin dolor vel urna rutrum mattis. Mauris nisi ligula, rhoncus eget risus blandit, rutrum laoreet lectus.

	filename	time	group_id
0	AMAR170.20240426T024445Z.wav	11.5	0
1	AMAR170.20240426T024445Z.wav	11.6	0
2	AMAR170.20240426T024445Z.wav	43.1	1
3	AMAR170.20240426T024445Z.wav	43.5	1
4	AMAR170.20240426T024445Z.wav	52.9	2
...
239	AMAR170.20240426T024445Z.wav	554.6	119
240	AMAR170.20240426T024445Z.wav	567.4	120
241	AMAR170.20240426T024445Z.wav	567.5	120
242	AMAR170.20240426T024445Z.wav	592.0	121
243	AMAR170.20240426T024445Z.wav	592.4	121

244 rows × 3 columns

Conclusão

- Desenvolvemos um detector automático de assobios de golfinho utilizando Python e aprendizado de máquina.
- O uso dos coeficientes de Mel permitiu representar bem os fragmentos sonoros.
- O modelo LGBMClassifier apresentou um bom desempenho com nível de acurácia considerável.
- Reduzimos falsos positivos aplicando um filtro temporal após a classificação.
- A solução final torna o processo de detecção mais rápido, leve e escalável.
- O pipeline pode ser ajustado para outros tipos de eventos bioacústicos no futuro.