

SOCIEDADE EDUCACIONAL DE SANTA CATARINA – UNISOCIESC
CURSO GRADUAÇÃO BACHARELADO CIÊNCIAS DA
COMPUTAÇÃO

BRENO AUGUSTO COMANDOLLI – 152110382

CLASSIFICADOR DE GÊNERO MUSICAL

Desenvolvimento de um agente inteligente

BLUMENAU - SC

2024

BRENO AUGUSTO COMANDOLLI

CLASSIFICADOR DE GÊNERO MUSICAL

Desenvolvimento de um agente inteligente

Agente inteligente para realização
classificação de gênero musical,
apresentado a Universidade Unisociesc,
como parte das exigências para conclusão
da unidade curricular de Inteligência
artificial.

Professor: Randerson Melville

BLUMENAU - SC

2024

SUMÁRIO

1 INTRODUÇÃO	5
2 DEFINIÇÃO DO PROBLEMA E COLETA DE DADOS	5
2.1 Defina um problema específico para o agente resolver	5
2.2 Escolha um conjunto de dados adequado (pode usar datasets públicos como os disponíveis no Kaggle ou no Scikit-Learn)	5
3 ESCOLHA O MODELO INICIAL E PRÉ-PROCESSAMENTO	6
3.1 Escolha um modelo base para iniciar o desenvolvimento do agente (ex: árvore de decisão, random forest, KNN, ou uma rede neural simples)	6
3.2 Realize o pré-processamento dos dados, incluindo limpeza, normalização e divisão em conjuntos de treino e teste	7
4 TREINAMENTO DO AGENTE E AVALIAÇÃO INICIAL	17
4.1 Treine o modelo base com o conjunto de dados escolhido	17
4.2 Avalie a performance do modelo base usando métricas apropriadas ao problema (ex: acurácia, precisão, recall, F1-score, etc.)	18
4.3 Documente os resultados iniciais e analise os pontos fortes e fracos do modelo	19
5 APRIMORAMENTO DO MODELO	19
5.1 Após a análise dos resultados iniciais, identifique e implemente possíveis melhorias. Sugestões	19
6 IMPLEMENTAÇÃO DO AGENTE APRIMORADO E TREINAMENTO	25
6.1 Implemente as melhorias propostas e re-treine o agente com os novos ajustes	25
6.2 Compare os novos resultados com os do modelo base para verificar se houve melhorias significativas	32
7 VALIDAÇÃO E ANÁLISE FINAL	33
7.1 Avalie o desempenho do agente aprimorado com o conjunto de dados de teste	33
7.2 Documente a performance final e avalie se o agente atende ao objetivo inicial	33
7.3 Crie um conjunto de testes para demonstrar o funcionamento do agente em diferentes cenários	33

7 CONCLUSÃO	34
8 REFERÊNCIAS	35

1 INTRODUÇÃO

Esse projeto tem o objetivo de desenvolver um agente inteligente para classificação de gênero musical, pois através desse estudo, desenvolvimento e melhoria do agente será possível aprender a utilizar múltiplas técnicas no processo de criação de algoritmos de inteligência artificial. O agente deverá conseguir distinguir dois gêneros diferentes, ou seja, classificação booleana. Dessa forma, o projeto terá um pré-processamento da base de dados utilizada, realização do melhor tipo de modelo de classificação de dados para esse cenário (o que apresentou melhor performance) e por fim os ajustes de hiper parâmetros, para melhorar a assertividade e ter um modelo com um percentual bom de classificação. Ademais, o relatório a seguir mostra como exatamente foi realizado o processo para chegar em um modelo de classificação de gêneros musicais bom.

2 DEFINIÇÃO DO PROBLEMA E COLETA DE DADOS

2.1 Defina um problema específico para o agente resolver

O agente deverá poder classificar os gêneros de músicas como pop ou rock, ou seja, ao informar dados detalhados sobre músicas rock ou pop, o agente deverá poder indicar qual é o gênero da música indicada. Observação: nesse contexto o agente só deverá receber músicas desses dois tipos de gênero para poder classificar entre um ou outro gênero. Porém, a evolução futura do agente é naturalmente poder classificar mais gêneros musicais, não deixando músicas desassistidas. Mas o foco desse projeto é classificar músicas do tipo pop ou rock, conforme classificação do modelo classificador de gêneros, que observa através de dados musicais detalhados da estrutura da música. Além disso, esta escolha foi tomada, devido a limitação dos dados da base, que causavam um desempenho significativamente menor, devido a distribuição dos dados está muito desigual entre os gêneros, isso será melhor abordado durante o relatório.

2.2 Escolha um conjunto de dados adequado (pode usar datasets públicos como os disponíveis no Kaggle ou no Scikit-Learn)

Após testar algumas bases distintas, percebi que a correção entre as colunas estava sempre dando baixa na maioria das bases, pois ao realizar avaliações da coleção, como a matriz

de correlação, quase sempre apresentava que as colunas de treinamento não impactavam muito no gênero que é a coluna alvo/classe, uma vez que, bases dedicadas a informações gerais das músicas, não são voltadas/filtradas para a classificação de gênero, ou seja, dados que são mais interessantes para a classificação de gênero musical, além da baixa correlação, foi possível observar que, nos treinamentos iniciais mostravam acurácias baixas, em volta dos 40, fazendo várias confusões na matriz de confusão, como por exemplos falsos positivos e falsos negativos, por esse motivo, tentei consultar uma Api para realizar a criação de um dataset que fosse feito para trazer somente as informações relevantes para a classificação, porém a api que tentei usar do Spotify, foi recentemente bloqueada/descontinuada pois estava sendo usada para pirataria. Todavia, encontrei uma base dedicada a classificação de gênero, onde os dados eram totalmente voltados para realização de classificação de gêneros, o que facilitou o desenvolvimento do agente inteligente. Coleção de dados escolhida para treinamento:

- <https://www.kaggle.com/datasets/purumalgi/music-genre-classification>

3 ESCOLHA O MODELO INICIAL E PRÉ-PROCESSAMENTO

3.1 Escolha um modelo base para iniciar o desenvolvimento do agente (ex: árvore de decisão, random forest, KNN, ou uma rede neural simples)

Foi escolhido como modelo inicial o **Random Forest**, devido a sua robustez e capacidade de lidar eficientemente com dados estruturados e complexos. Ele se destaca por algumas características importantes como:

- Redução de overfitting: diferentemente de uma única árvore de decisão, que pode se ajustar excessivamente aos dados de treinamento, o Random Forest combina previsões de várias árvores independentes;
- Robustez e desempenho: é mais estável e mais confiável por ter várias árvores (ensemble), já que, consegue capturar relações complexas entre variáveis sem depender muito de qualquer uma delas;
- Flexibilidade nas escolhas de variáveis: KNN não se comporta bem com escalar diferentes de variáveis, uma vez que se baseia na distancia dos elementos;

- Capacidade de lidar com grandes quantidades de dados: É um modelo que consegue se sair melhor com grandes quantidades de dados, extraindo insights que são importantes, especialmente quando há muitas variáveis;
- Interpretação e importância das variáveis: esse algoritmo também oferece vantagem para calcular a importância de cada variável, ajudando a identificar quais características tem um maior impacto na classificação.

3.2 Realize o pré-processamento dos dados, incluindo limpeza, normalização e divisão em conjuntos de treino e teste

Primeiramente, foi feito o carregamento da base de dados de treino obtidos no dataset escolhido anteriormente:

Imagem 1 – Leitura dos dados:

```
[ ] #Carregamento do dataset para treinamento de gênero
train_df = pd.read_csv('train.csv')

#Quantidade linhas e colunas
train_df.shape

(17996, 17)
```

Em seguida, foi impresso algumas estatísticas importantes para ajudar na identificação das melhorias dos dados no pré-processamento:

Imagem 2 – Exibição dos dados do dataset:

```
#Visualização do dataset
train_df.head()
```

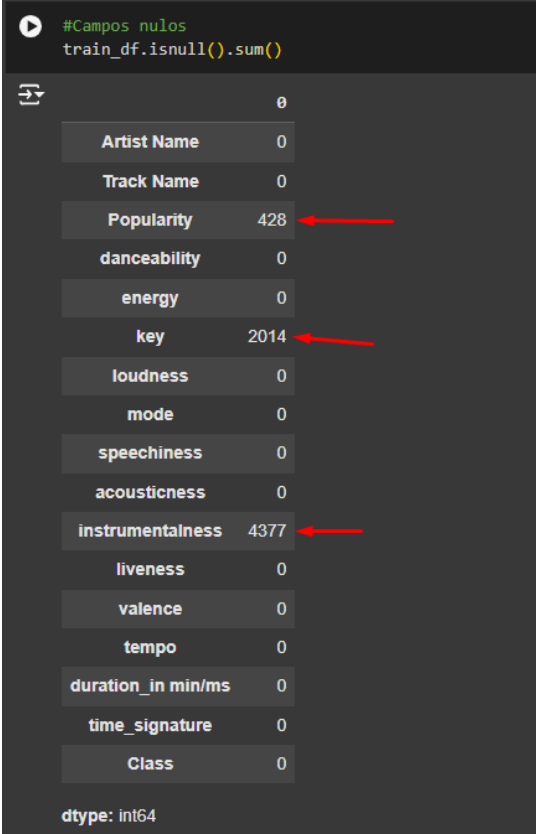
	Artist Name	Track Name	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_in min/ms	time_signature	Class
0	Bruno Mars	That's What I Like (feat. Silk Sonic)	60.0	0.854	0.564	1.0	-4.964	1	0.0485	0.017100	NaN	0.0849	0.8990	134.071	234596.0	4	5
1	Boston	Hitch a Ride	54.0	0.382	0.814	3.0	-7.230	1	0.0406	0.001100	0.004010	0.1010	0.5690	116.454	251733.0	4	10
2	The Ramocasts	No Side to Fall In	35.0	0.434	0.614	6.0	-8.334	1	0.0525	0.486000	0.000196	0.3940	0.7870	147.681	109667.0	4	6
3	Deno	Lingo (feat. J.I & Chunkz)	66.0	0.853	0.597	10.0	-6.528	0	0.0555	0.021200	NaN	0.1220	0.5690	107.033	173968.0	4	5
4	Red Hot Chili Peppers	Nobody Weird Like Me - Remastered	53.0	0.167	0.975	2.0	-4.279	1	0.2160	0.000169	0.016100	0.1720	0.0918	199.060	229960.0	4	10

```
[ ] #Informações do dataset
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17996 entries, 0 to 17995
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Artist Name           17996 non-null object
1   Track Name            17996 non-null object
2   Popularity             17568 non-null float64
3   danceability           17996 non-null float64
4   energy                 17996 non-null float64
5   key                   15982 non-null float64
6   loudness              17996 non-null float64
7   mode                  17996 non-null int64
8   speechiness           17996 non-null float64
9   acousticness          17996 non-null float64
10  instrumentalness       13619 non-null float64
11  liveness              17996 non-null float64
12  valence                17996 non-null float64
13  tempo                 17996 non-null float64
14  duration_in min/ms    17996 non-null float64
15  time_signature         17996 non-null int64
16  Class                 17996 non-null int64
dtypes: float64(12), int64(3), object(2)
memory usage: 7.3+ MB
```

Para descobrir as colunas que tem valores em branco, foi feita a seguinte abordagem de exibição estatísticas:

Imagem 3 – Dados nulos do dataset:



The screenshot shows a Jupyter Notebook cell with the code `#Campos nulos` and `train_df.isnull().sum()`. The output is a table with two columns: the feature name and the count of null values. Three rows are highlighted with red arrows: 'Popularity' with 428 nulls, 'key' with 2014 nulls, and 'instrumentalness' with 4377 nulls. The data type is shown as 'dtype: int64' at the bottom.

	0
Artist Name	0
Track Name	0
Popularity	428
danceability	0
energy	0
key	2014
loudness	0
mode	0
speechiness	0
acousticness	0
instrumentalness	4377
liveness	0
valence	0
tempo	0
duration_in min/ms	0
time_signature	0
Class	0

dtype: int64

Através desses dados, foi possível perceber que três colunas têm dados não preenchidos, o que pode ser um problema para a classificação. Para escolher o que fazer com esse foi adotado o seguinte raciocínio para as colunas:

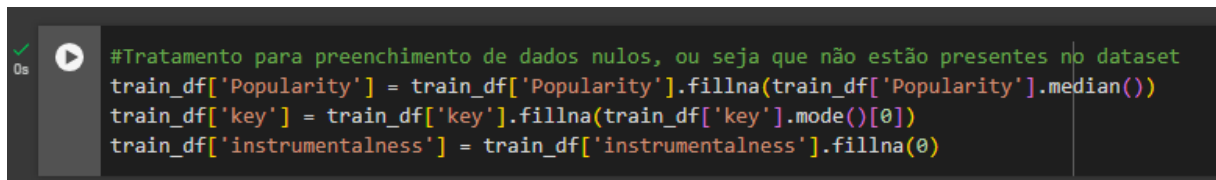
1º - Popularity: é uma variável que indica a média de popularidade das músicas, ou seja, através da interação dos usuários com a música, que vai aumentando a média de popularidade, para que a média de popularidade não seja afetada com os valores nulos, foi adotado a abordagem de preencher todos os valores em branco com a mediana dos valores preenchidos da coluna. Contudo, não foi usado a média, pois a mesma pode sofrer impacto de outliers;

2º - Key: é uma variável categórica, ou seja, representa uma categoria dentro de um conjunto de possibilidades, e está relacionado ao tom musical adotado na música, dessa forma, para casos como esse de variáveis categorias, é indicado colocar a moda da coluna, ou seja, qual é o valor que mais se repete no conjunto;

3º - Instrumentalness: é uma variável que indica o quando de instrumentalidade está presente na música, é representado por valores flutuantes que mostram o grau através do intervalo 0 até 1. Dessa forma, para não acabar gerando overfitting indicando que há instrumentalidade onde não tem, foi adicionado a substituição de vazio para 0, indicando que quando não tem informado o valor, significa que a musica não tem instrumentalidade significativa.

Código abaixo com a implementação dessas abordagens que tiveram seus desempenhos testados posteriormente:

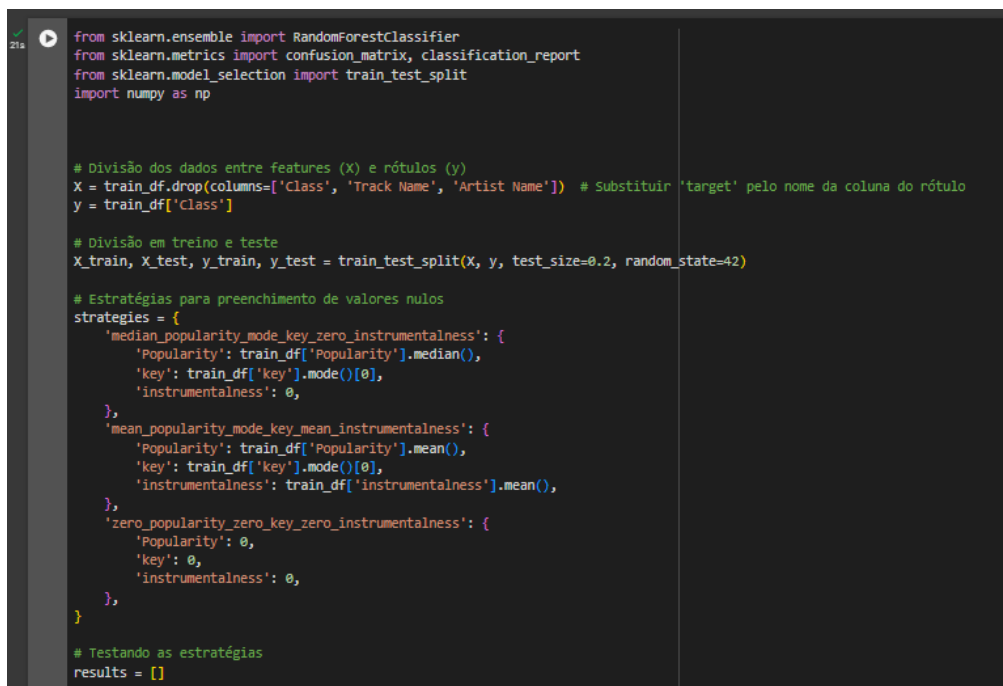
Imagem 4 – Tratamento de valores nulos:



```
#Tratamento para preenchimento de dados nulos, ou seja que não estão presentes no dataset
train_df['Popularity'] = train_df['Popularity'].fillna(train_df['Popularity'].median())
train_df['key'] = train_df['key'].fillna(train_df['key'].mode()[0])
train_df['instrumentalness'] = train_df['instrumentalness'].fillna(0)
```

Para garantir que foi a melhor abordagem, rodei um algoritmo que testou algumas combinações diferentes para os valores nulos. Foi possível concluir, que realmente essa foi a melhor abordagem, conforme imagens abaixo:

Imagem 5 – Algoritmo para validar melhores combinações de valores nulos (parte 1)



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import numpy as np

# Divisão dos dados entre features (X) e rótulos (y)
X = train_df.drop(columns=['Class', 'Track Name', 'Artist Name']) # Substituir 'target' pelo nome da coluna do rótulo
y = train_df['Class']

# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Estratégias para preenchimento de valores nulos
strategies = {
    'median_popularity_mode_key_zero_instrumentalness': {
        'Popularity': train_df['Popularity'].median(),
        'key': train_df['key'].mode()[0],
        'instrumentalness': 0,
    },
    'mean_popularity_mode_key_mean_instrumentalness': {
        'Popularity': train_df['Popularity'].mean(),
        'key': train_df['key'].mode()[0],
        'instrumentalness': train_df['instrumentalness'].mean(),
    },
    'zero_popularity_zero_key_zero_instrumentalness': {
        'Popularity': 0,
        'key': 0,
        'instrumentalness': 0,
    },
}

# Testando as estratégias
results = []
```

6 – Algoritmo para validar melhores combinações de valores nulos (parte 1)

```
for strategy_name, strategy_values in strategies.items():
    # Aplicando o preenchimento
    X_train_filled = X_train.copy()
    X_test_filled = X_test.copy()

    for col, value in strategy_values.items():
        X_train_filled[col] = X_train_filled[col].fillna(value)
        X_test_filled[col] = X_test_filled[col].fillna(value)

    # Treinando o modelo Random Forest
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train_filled, y_train)

    # Predição no conjunto de teste
    y_pred = model.predict(X_test_filled)

    # Avaliação do modelo
    cm = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    accuracy = report['accuracy']

    results.append({
        'strategy': strategy_name,
        'confusion_matrix': cm,
        'accuracy': accuracy,
        'classification_report': report,
    })

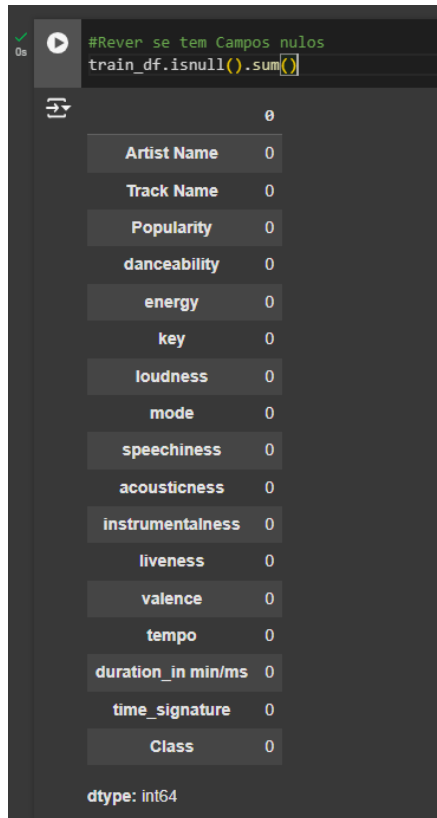
# Exibindo os resultados
for result in results:
    print(f"Strategy: {result['strategy']}")
    print("Confusion Matrix:")
    print(result['confusion_matrix'])
    print(f"Accuracy: {result['accuracy']}")
    print("Classification Report:")
    print(result['classification_report'])
    print("-" * 50)
```

7 – Algoritmo para validar melhores combinações de valores nulos (resultados)

```
Strategy: median_popularity_mode_key_zero_instrumentalness
Confusion Matrix:
[[113  0  0  5  8  0  1  3  0  5  1]
 [  0  9 14  0  0  9 10  0  4 15 12]
 [  0  0 10  0  0  2 30  0  0 29 11]
 [  7  0  0 62  3  0  0  5  0  1  0]
 [15  0  0  0 50  0  0  1  0  4  1]
 [  0  2  3  0  0 17 14  0  0 46 19]
 [  0 73 11  0  0 11 13  0 13 59 19]
 [  6  0  0  1  1  0  0 95  0  0  0]
 [  0  0  0  0  0  0 11  0 210 1 16]
 [  7  8 11  1  6 40 56  0 2 282 11]
 [  6 53 36  2  6  8 180  0 94 83 582]]
Accuracy: 0.5072222222222222
Classification Report:
{'0': {'precision': 0.7337662337662337, 'recall': 0.8308823529411765, 'f1-score': 0.7793103448275862, 'support': 136.0},
-----}
Strategy: mean_popularity_mode_key_mean_instrumentalness
Confusion Matrix:
[[108  0  0  7 11  0  1  5  0  3  1]
 [  0  7 10  0  0  8 10  0  4 19 13]
 [  0  1 11  0  0  4 27  0  2 21 11]
 [10  0  0 61  3  0  0  2  0  2  0]
 [15  0  0  0 52  0  0  0  0  3  1]
 [  0  2  6  0  0 18  7  0  0 46 15]
 [  0 71 12  0  0 20 12  0  9 60 20]
 [  6  0  0  1  1  0  0 95  0  0  0]
 [  0  0  0  0  0  1 13  0 20  2 16]
 [  8  6  8  2  6 41 55  1 2 281 12]
 [  4 53 37  3  8 10 101  1 99 84 570]]
Accuracy: 0.5005555555555555
Classification Report:
{'0': {'precision': 0.7152317880794702, 'recall': 0.7941176470588235, 'f1-score': 0.7526132404181185, 'support': 136.0},
-----}
Strategy: zero_popularity_zero_key_zero_instrumentalness
Confusion Matrix:
[[108  0  0  9 10  0  0  2  0  4  3]
 [  0  9 10  0  0 10 10  0  5 16 12]
 [  0  0 10  0  0  3 28  0  1 33 11]
 [  7  0  0 61  3  0  0  5  0  2  0]
 [11  0  0  0 52  0  0  2  0  5  1]
 [  0  1  2  0  0 17 21  0  0 42 18]
 [  0 68 11  0  0 15 14  0 13 54 19]
 [  6  0  0  3  0  0  0 94  0  0  0]
 [  0  0  0  0  0  0 11  0 20  1 16]
 [  5  5  8  2  6 41 58  1 2 283 12]
 [  6 51 35  3 10  8 97  0 93 86 581]]
Accuracy: 0.5038888888888889
Classification Report:
{'0': {'precision': 0.7552447552447552, 'recall': 0.7941176470588235, 'f1-score': 0.7741935483870968, 'support': 136.0},
-----}
```

Após esse tratamento, rodei as estatísticas de campos nulos novamente para garantir que não haviam mais valores nulos:

Imagem 8 – Tratamento de valores nulos:



```
#Ver se tem Campos nulos
train_df.isnull().sum()
```

	0
Artist Name	0
Track Name	0
Popularity	0
danceability	0
energy	0
key	0
loudness	0
mode	0
speechiness	0
acousticness	0
instrumentalness	0
liveness	0
valence	0
tempo	0
duration_in min/ms	0
time_signature	0
Class	0

dtype: int64

Outro ponto válido na realização dos tratamentos de dados é a remoção de colunas que podem causar algum impacto negativo, ou que não fazem sentido ser usadas na classificação, abaixo o detalhamento de cada uma:

1º - Popularity: é uma coluna que ao ser deixada na classificação pode acabar causando viés na base de dados, ou seja, tendenciar a base a ir para uma linha que não deveria, devido a distorção dos dados. Uma vez que, a popularidade da música não deveria influenciar qual é o gênero, mas sim dados característicos da montagem da música, popularidade é algo que está relacionado ao gosto musical das pessoas;

2º - Duration in min/ms: O tempo da música é a mesma coisa, pode mudar conforme o passar dos anos, de um artista para outro, e de uma letra para outra, sendo que o gênero permanece o mesmo, manter ela na classificação pode influenciar indevidamente a classificação;

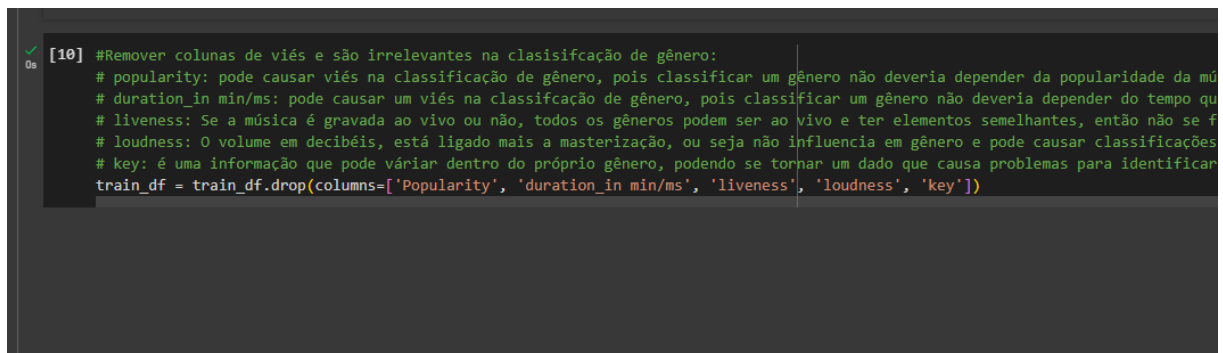
3º - Liveness: novamente é um dado que independe do gênero, indica o quando de características de música ao vivo tem na música, podendo criar vícios no treinamento da IA;

4° - Loudness: é uma variável que indica o volume em decibéis e por isso não influencia na classificação do gênero, mas sim na qualidade e volume da música;

5° - Key: é uma informação que pode variar dentro do próprio gênero música, embora possa ter seu grau de importância para subgênero ou algo semelhante, não tem um impacto muito grande na classificação (isso será melhor mostrado na matriz de correlação das variáveis)

Embora a remoção dessas variáveis causou um pouco de impacto na acurácia e em números de confusão, recall, precisão e f1 score, foi decidido manter pois a influência delas no modelo foi classificado como perigoso, pois pode causar vieses no modelo, ou seja, nem sempre acurácia alta significa que o modelo está melhor, e nesse caso a influência dessas variáveis comprovam isso, já no caso da variável key, não tem uma influência muito grande o que torna a variável dispensável no modelo atual.

Imagem 9 – Remoção de campos indevidos do dataset:



```
[10] #Remover colunas de viés e são irrelevantes na classificação de gênero:
# popularity: pode causar viés na classificação de gênero, pois classificar um gênero não deveria depender da popularidade da música
# duration_in min/ms: pode causar um viés na classificação de gênero, pois classificar um gênero não deveria depender do tempo que a música foi gravada
# liveness: Se a música é gravada ao vivo ou não, todos os gêneros podem ser ao vivo e ter elementos semelhantes, então não se faz sentido classificar por liveness
# loudness: O volume em decibéis, está ligado mais a masterização, ou seja não influencia em gênero e pode causar classificações erradas
# key: é uma informação que pode variar dentro do próprio gênero, podendo se tornar um dado que causa problemas para identificar o gênero
train_df = train_df.drop(columns=['Popularity', 'duration_in min/ms', 'liveness', 'loudness', 'key'])
```

Abaixo mostra o teste realizado para verificar o impacto que essas remoções fizeram no treinamento da IA, é possível perceber que a medida que vai removendo as colunas vai criando um certo impacto negativo, o que é um impacto negativo que faz sentido, pois popularidade, músicas ao vivo e tempo da música, realmente podem impactar a classificação de gênero, mas mantê-las é um problema, pois induz a IA a ir para uma linha que não é apenas classificação de dados musicais, mas sim contextualizações de cenários onde a música está inserida, pode ser válido para outros modelos, mas para um modelo que tem o objetivo de não se importar com a opinião do público, ou tempo da música, não faz sentido manter. Finalmente, é possível concluir que no final das contas mesmo com o impacto, removendo todas essas colunas a acurácia ficou bem semelhante:

Imagem 10 – Remoção de campos indevidos do dataset:

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

X = train_df.drop(columns=['Class', 'Track Name', 'Artist Name'])
y = train_df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def train_and_evaluate(X_train, X_test, y_train, y_test):
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)

    return accuracy, cm, report

print("Modelo com todas as colunas:")
accuracy, cm, report = train_and_evaluate(X_train, X_test, y_train, y_test)
print(f"Acurácia: {accuracy}")
print("Matriz de Confusão:")
print(cm)
print("Relatório de Classificação:")
print(report)
print("=" * 50)

columns_to_remove = ['Popularity', 'duration_in min/ms', 'liveness', 'loudness', 'key']

for column in columns_to_remove:
    print(f"Removendo coluna: {column}")

    X_train_dropped = X_train.drop(columns=[column])
    X_test_dropped = X_test.drop(columns=[column])

    # Avaliação do modelo
    accuracy, cm, report = train_and_evaluate(X_train_dropped, X_test_dropped, y_train, y_test)
    print(f"Acurácia: {accuracy}")
    print("Matriz de Confusão:")
    print(cm)
    print("Relatório de Classificação:")
    print(report)
    print("=" * 50)

```

Imagem 11 – Remoção de campos indevidos do dataset (resultado parte 1)

```

print("=" * 50)

Modelo com todas as colunas:
Acurácia: 0.5072222222222222
Matriz de Confusão:
[[113  0  0  5  8  0  1  3  0  5  1]
 [  0  9 14  0  0  9 107  0  4 15 128]
 [  0  0 108  0  0  2  30  0  0 29 112]
 [  7  0  0 62  3  0  0  5  0  1  0]
 [15  0  0  0 50  0  0  1  0  4  1]
 [  0  2  3  0  0 178 14  0  0 46 19]
 [  0 73 11  0  0 11 137  0 13 59 196]
 [  6  0  0  1  1  0  0 95  0  0  0]
 [  0  0  0  0  0  0 11  0 210 1 160]
 [  7  8 11  1  6 40 56  0 2 282 118]
 [  6 53 36  2  6  8 100  0 94 83 582]]
Relatório de Classificação:
{'0': {'precision': 0.7337662337662337, 'recall': 0.8308823529411765, 'f1-score': 0.7793103448275862, 'support': 136.0}, '1':
=====
Removendo coluna: Popularity
Acurácia: 0.4847222222222222
Matriz de Confusão:
[[104  0  0 12  8  0  1  2  0  6  3]
 [  0  3 10  0  0  9 117  0  4 15 120]
 [  0  0 80  0  0  3  30  0  1 34 133]
 [  7  0  0 59  2  0  0  6  0  2  2]
 [18  0  0  0 42  0  0  1  0 10  0]
 [  0  1  5  0  0 175 18  0  0 44 19]
 [  0 68  8  0  0 12 141  0 12 60 199]
 [  6  0  0  3  0  0  0 94  0  0  0]
 [  0  0  0  0  0  0  9 210  0 163]
 [  7  7  7  6 18 44 60  0 2 262 118]
 [  9 52 40  4  5  9 35  0 95 86 575]]
Relatório de Classificação:
{'0': {'precision': 0.6887417218543046, 'recall': 0.7647058823529411, 'f1-score': 0.7247386759581882, 'support': 136.0}, '1':
=====
Removendo coluna: duration_in min/ms
Acurácia: 0.43
Matriz de Confusão:
[[ 25  3 14  2  0  1 22  0  0 33 36]
 [  7  3  9  1  3 11 100  9  6 16 121]
 [  3  1 92  7  2  3 26  4  0 40 103]
 [  1  0 11 29  1  7  8  2  0  8 111]
 [  5  0  2  0  5  1  0  1  0 14 43]
 [  0  1  1  0  0 182 15  0  0 51 12]
 [  8 69 12  2  1 11 116 12 13 60 196]
 [  2  0  4  0  0  1  9 87  0  0  0]
 [  0  0  0  0  0  0 12 1 197 1 171]
 [13  6 12  4  2 49 53  0 1 257 134]
 [  4 51 39  4  5  9 97  2 103 101 555]]
Relatório de Classificação:
{'0': {'precision': 0.36764705882352944, 'recall': 0.10382352941176472, 'f1-score': 0.24509083921568626, 'support': 136.0},
=====

```

Imagem 12 – Remoção de campos indevidos do dataset (resultado parte 2)

```

C0: {'precision': 0.7067638033234, 'recall': 0.7103233234176742, 'f1-score': 0.70850521308026, 'support': 136.0}
=====
Removendo coluna: liveness
Acurácia: 0.5030555555555556
Matriz de Confusão:
[[112  0  0  9  7  0  0  2  0  5  1]
 [  0  5 11  0  0 11 110  0  5 17 127]
 [  0  0 109  0  0  3 29  0  2 31 107]
 [11  0  0 59  3  0  0  3  0  2  0]
 [10  0  0  0 54  0  0  2  0  4  1]
 [  0  2  1  0  0 179 14  0  0 48 18]
 [  0 67 14  0  0 14 150  0 12 49 194]
 [  6  0  0  1  1  0  0 95  0  0  0]
 [  0  0  0  0  0  0 10  0 209 1 162]
 [10  7 14  3  6 40 53  0  2 272 124]
 [  6 52 41  3  7  7 107  1 96 83 567]]
Relatório de Classificação:
{'0': {'precision': 0.7225806451612903, 'recall': 0.8235294117647058, 'f1-score': 0.7697594501718213, 'support': 136.0},
=====
Removendo coluna: loudness
Acurácia: 0.49277777777777776
Matriz de Confusão:
[[116  0  0  5  8  0  1  0  0  5  1]
 [  0  4 12  0  0 10 111  0  5 17 127]
 [  0  0 111  0  0  5 27  0  0 34 104]
 [  7  0  0 60  3  0  0  3  0  5  0]
 [15  0  0  0 52  0  0  0  0  3  1]
 [  0  1  3  0  0 179 15  0  0 48 16]
 [  0 69 17  0  0 15 138  0 14 48 199]
 [  7  0  0  2  0  0  0 94  0  0  0]
 [  0  0  0  0  0  0 17  0 202 1 162]
 [  8  6 11  2  9 42 62  0  1 272 118]
 [  6 52 37  1  9  9 113  0 100 97 546]]
Relatório de Classificação:
{'0': {'precision': 0.7295597484276729, 'recall': 0.8529411764705882, 'f1-score': 0.7864406779661017, 'support': 136.0},
=====
Removendo coluna: key
Acurácia: 0.5080555555555556
Matriz de Confusão:
[[114  0  0  8  8  0  0  1  0  4  1]
 [  0  9 12  0  0 10 106  0  6 15 128]
 [  0  1 107  0  0  3 26  0  0 29 115]
 [  9  0  0 63  2  0  0  3  0  1  0]
 [12  0  0  0 54  0  0  2  0  2  1]
 [  0  1  3  0  0 186 15  0  0 36 21]
 [  0 72 10  0  0 15 142  0 10 54 197]
 [  5  0  0  1  1  0  0 96  0  0  0]
 [  0  0  0  0  0  0 12  0 208 1 161]
 [  7  5  7  2  5 44 57  0  1 274 129]
 [  5 53 31  3 11  8 102  0 94 87 576]]
Relatório de Classificação:
{'0': {'precision': 0.75, 'recall': 0.8382352941176471, 'f1-score': 0.7916666666666666, 'support': 136.0}, '1': {'precisi
=====

```

Verificar a correlação das variáveis foi o próximo ponto a ser abordado, para ver se realmente fazia sentido manter algumas colunas na classificação, ou não. Além disso, verificar qual é as colunas com maior influencia com gênero. Abaixo o código utilizado:

Imagem 13 – Código da matriz confusão que mostra a correlação das colunas:

```

from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# Passo 1: Converter colunas categóricas para numéricas usando Label Encoding
label_encoders = {}
for col in train_df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    label_encoders[col] = le

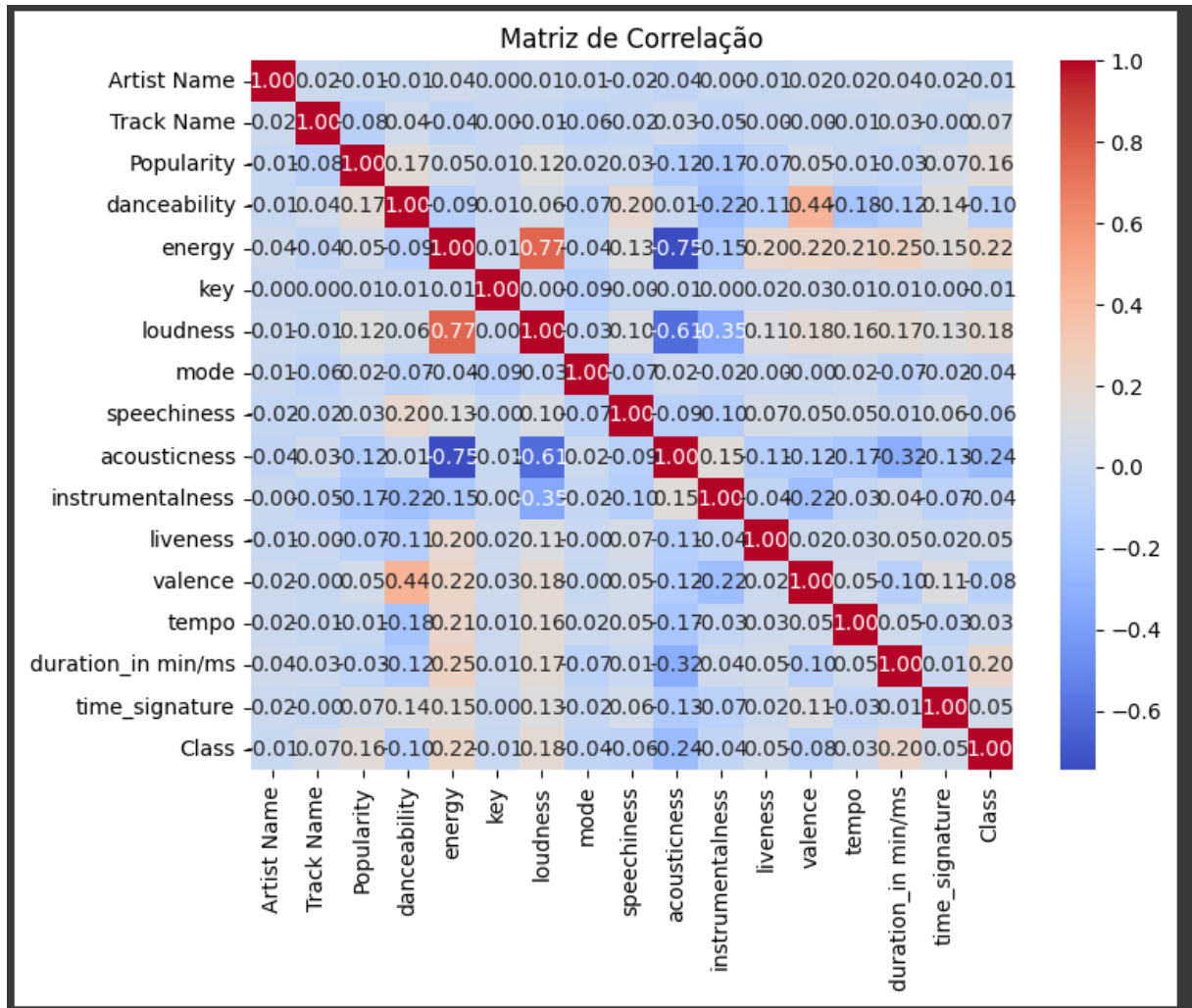
# Passo 2: Calcular a matriz de correlação
correlation_matrix = train_df.corr()

# Exibir a matriz de correlação
print("Matriz de Correlação:")
print(correlation_matrix)

# Passo 3: Visualizar a matriz de correlação com um heatmap (opcional)
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matriz de Correlação")
plt.show()

```

Imagem 14 – Plotagem da matriz confusão que mostra a correlação das colunas:



Na matriz é possível perceber que a correlação das colunas com “Class” é relativamente baixa. Ademais, é possível observar que colunas de viés são altas como a popularidade e o tempo da música, o que poderia ser um problema lá para frente, além disso, que key realmente tem uma relação bem baixa com a class e por realmente não fazer muito sentido no modelo foi retirado conforme relatos e operações anteriores. As demais colunas embora com sua importância baixa, em conjunto fazem ser possível classificar o gênero, conforme mais a frente será indicada.

Na sequência, foi feito um tratamento para converter os dados de texto em numéricos, para poder utilizar os textos na classificação, esse processo é chamado Label Encoding, abaixo o código do tratamento:

Imagem 15 – Label Encoding para os campos de nome:

```

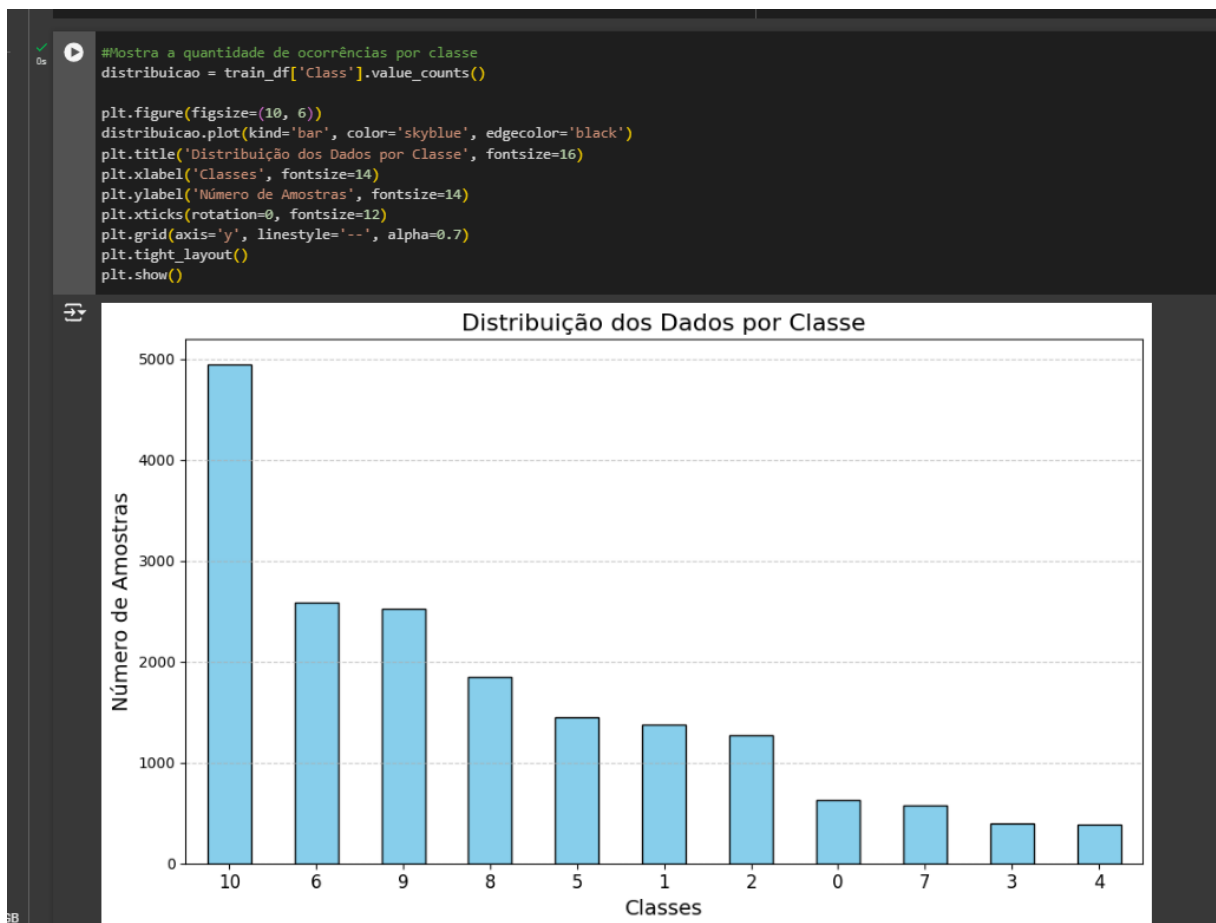
label_encoder_track = LabelEncoder()
label_encoder_artist = LabelEncoder()

# Aplicar o Label Encoding nas colunas
train_df['Track Name'] = label_encoder_track.fit_transform(train_df['Track Name'])
train_df['Artist Name'] = label_encoder_artist.fit_transform(train_df['Artist Name'])

```

Após esse tratamento, foi observado a distribuição dos dados da base para verificar quais gêneros tinham mais informações para conseguir classificar, abaixo código e gráfico de distribuição:

Imagem 16 – Gráfico de distribuição dos dados da coleção de treinamento da IA:



Após essa amostragem, foi simplificado o modelo para classificar apenas dois gêneros, esses 9 e 10, representados como pop e rock, assim, simplificando uma primeira versão do modelo, que posteriormente com mais dados, poderia expandir classificando mais gêneros. Conforme visto anteriormente, isso é justificável devido ao pré-processamento já ter sido feito e a acurácia está muito baixa, sem falar que a matriz confusão desses dois gêneros tiveram

bastante acertos o que ajudou na escolha das duas como alvo primário. Para filtrar a base foi utilizado o seguinte código:

Imagem 16 – Filtragem do dataset para só ficar com dados de pop e rock:

```
#Filtra o modelo para só pop e rock
train_df = train_df[train_df['Class'].isin([9, 10])]
```

4 TREINAMENTO DO AGENTE E AVALIAÇÃO INICIAL

4.1 Treine o modelo base com o conjunto de dados escolhido

O treinamento da IA foi feito conforme imagens abaixo, onde foi dividido os dados em 80% para treino e 20% para testes, além de não passar nenhum hiper parâmetro, e na sequência foi mostrado os relatórios de desempenho da IA:

Imagem 17 – Código de treino inicial:

```
X = train_df.drop(columns=['Class'])
y = train_df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Acurácia
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia: {accuracy:.2f}")

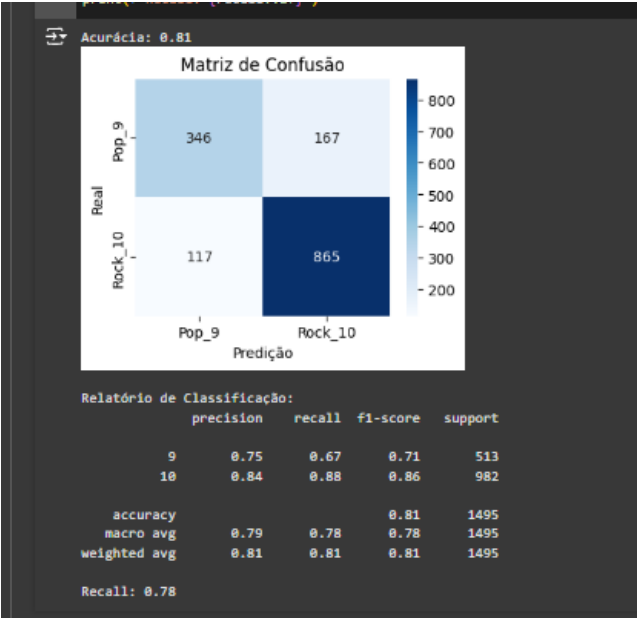
#Matriz de Confusão
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_9', 'Rock_10'], yticklabels=['Pop_9', 'Rock_10'])
plt.title('Matriz de Confusão')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()

#Relatório
print("\nRelatório de Classificação:")
print(classification_report(y_test, y_pred))

#Recall
recall = recall_score(y_test, y_pred, average='macro')
print(f"Recall: {recall:.2f}")
```

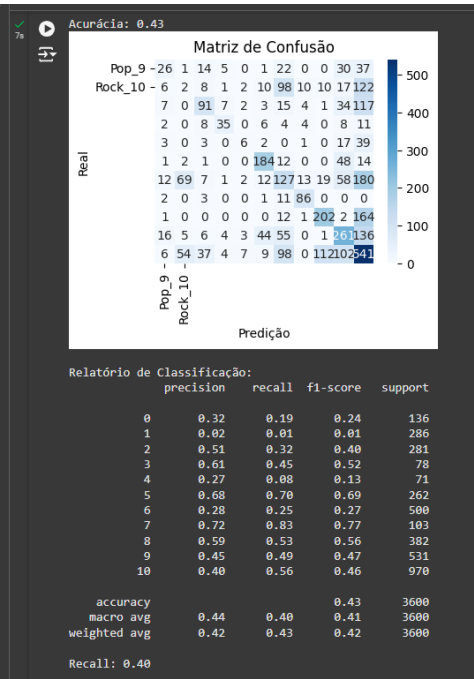
4.2 Avalie a performance do modelo base usando métricas apropriadas ao problema (ex: acurácia, precisão, recall, F1-score, etc.)

Imagem 18 – Código de treino inicial (resultados):



É possível destacar que teve um desempenho razoável, porém métricas como precision, recall e f1-score foram baixos para o gênero pop, enquanto do gênero rock foram razoáveis. Quando não filtrado os gêneros e treinados todos os gêneros a disparidade é muito grande:

Imagem 19 – Treinamento sem filtragem de gênero (motivo da escolha de filtrar):



Nesse modelo, é possível ver que a base está mais propícia a classificar determinados gêneros. Além disso, é um ponto de observação ver que há muitas disparidades na classificação dos gêneros, tendo dados de recall, precision, f1-score e confusões muito ruins, e outras intermediárias. Aqui é finalizado o relatório intermediário do projeto.

4.3 Documente os resultados iniciais e analise os pontos fortes e fracos do modelo

Basicamente, é possível concluir que o forte do modelo é a classificação de rock que tem todas as suas métricas razoavelmente boas, pop ainda não está com dados muito bons de recall, ou seja, identificou corretamente um número baixo de instâncias boas, além disso, a precisão e f1-score, que são instâncias positivas que realmente eram positivas e o equilíbrio entre precisão e recall respectivamente, também não estão altos.

5 APRIMORAMENTO DO MODELO

5.1 Após a análise dos resultados iniciais, identifique e implemente possíveis melhorias. Sugestões

a) Experimentar outros algoritmos de aprendizado:

Foi tentado executar a classificação de três tipos diferentes de modelos de classificação, que foram os três aprendidos em sala de aula. Porém o que se comportou melhor foi o Random Forest, mais para frente os modelos descartados voltaram a cena novamente para serem testados em conjunto em um meta-modelo, que será abordado depois. Abaixo código que mostra a diferença dos modelos de classificação em termos de desempenho:

Imagem 20 – Múltiplos treinamento para avaliar classificadores:

```

models = {
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42)
}

# Treinamento, predição e avaliação para cada modelo
for model_name, model in models.items():
    print(f"Treinando modelo: {model_name}")

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Acurácia
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Acurácia: {accuracy:.2f}")

    # Recall
    recall = recall_score(y_test, y_pred, average='macro')
    print(f"Recall: {recall:.2f}")

    # Relatório de Classificação
    print("\nRelatório de Classificação:")
    print(classification_report(y_test, y_pred))

    # Matriz de Confusão
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_9', 'Rock_10'], yticklabels=['Pop_9', 'Rock_10'])
    plt.title(f'Matriz de Confusão - {model_name}')
    plt.xlabel('Predição')
    plt.ylabel('Real')
    plt.show()
    print("\n" + "="*50 + "\n")

```

Imagem 21 – Múltiplos treinamento para avaliar classificadores (KNN):

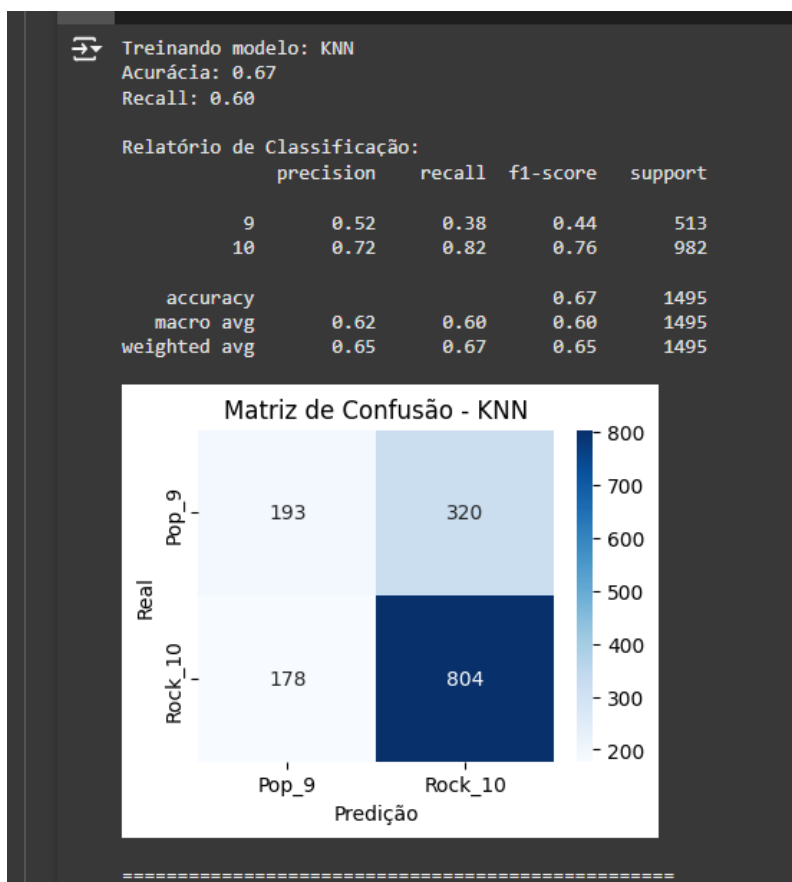


Imagem 22 – Múltiplos treinamento para avaliar classificadores (Decision Tree):

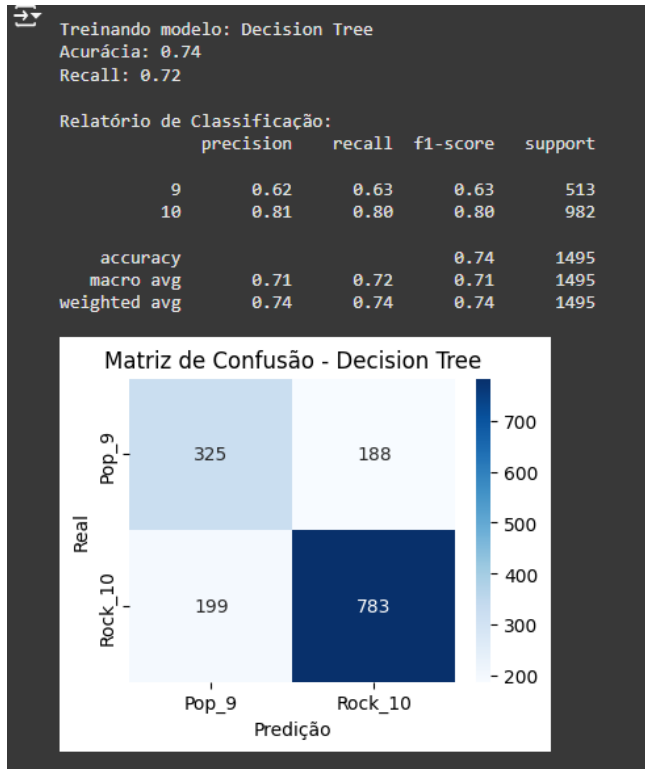
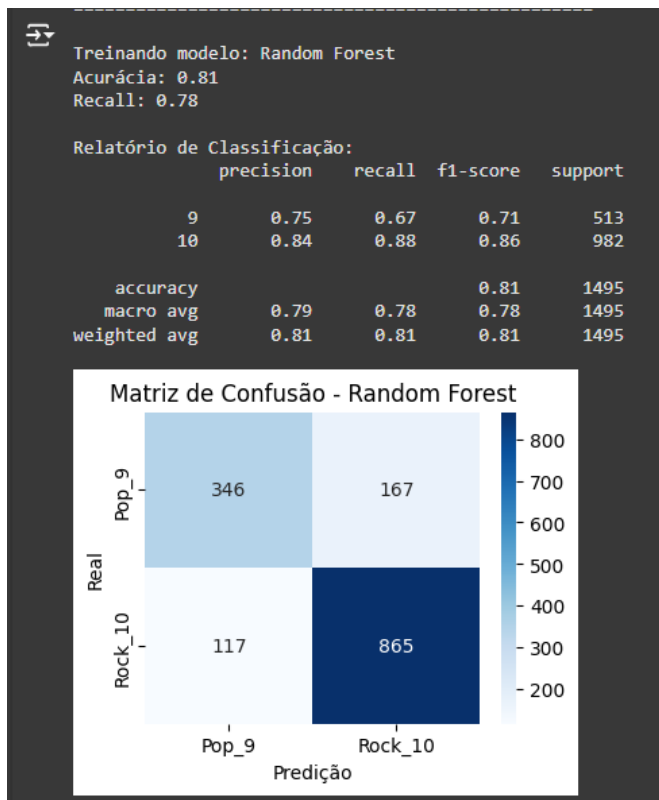


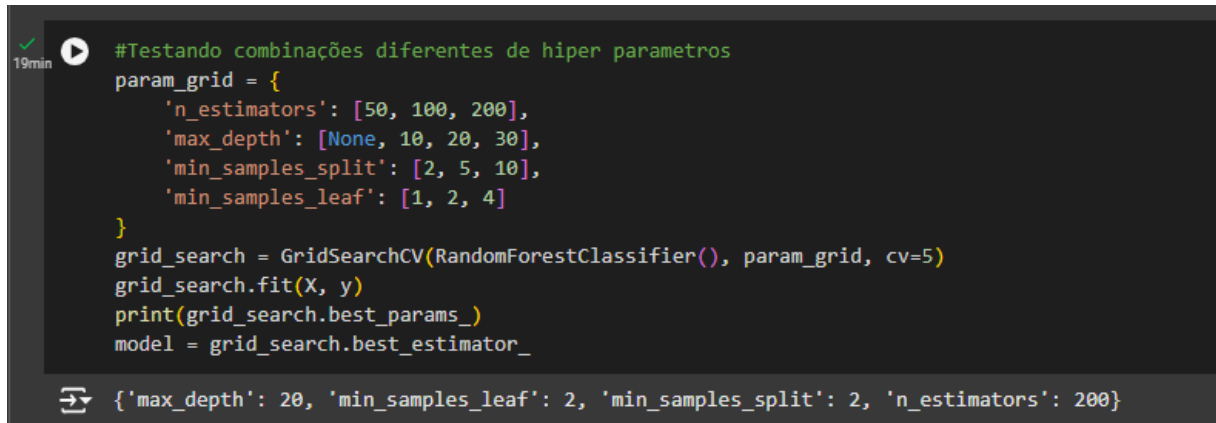
Imagem 23 – Múltiplos treinamento para avaliar classificadores (Random Forest):



b) Ajustar hiper parâmetros com técnicas como Grid Search ou Random Search:

Usando validação cruzada e uma lista de parâmetros, foi feito o código abaixo, para tentar achar a melhor combinação possível para o tipo de modelo escolhido:

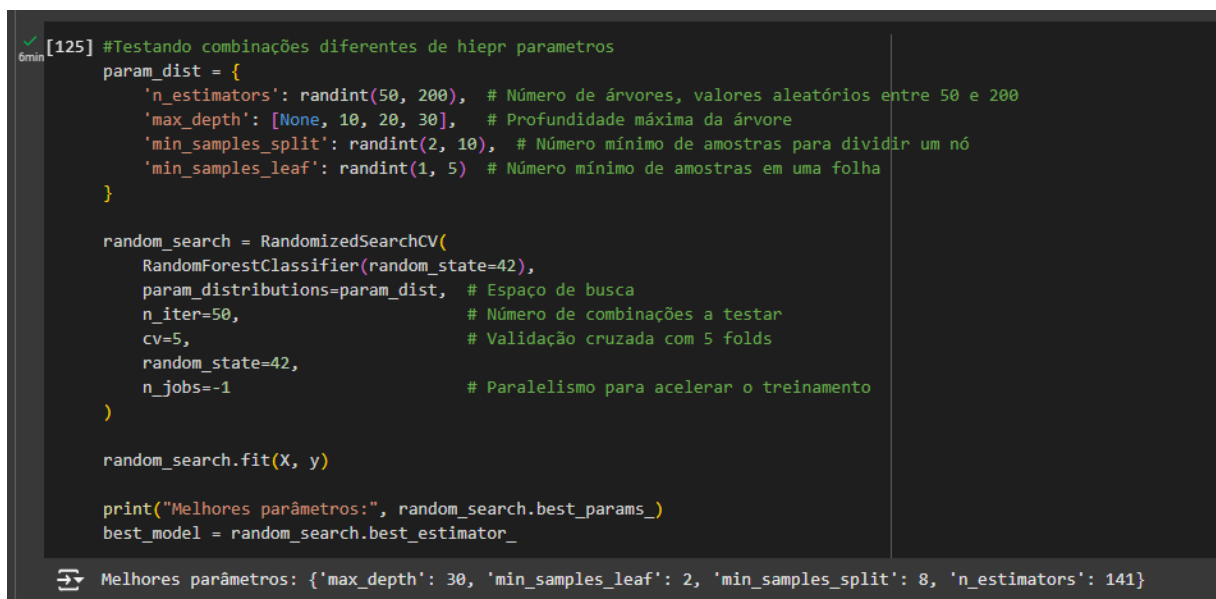
Imagem 24 – Código melhor combinação de hiper parâmetros usando Grid Search:



```
#Testando combinações diferentes de hiper parametros
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X, y)
print(grid_search.best_params_)
model = grid_search.best_estimator_

{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
```

Imagem 25 – Código melhor combinação de hiper parâmetros usando Random Search:



```
[125] #Testando combinações diferentes de hiper parametros
param_dist = {
    'n_estimators': randint(50, 200), # Número de árvores, valores aleatórios entre 50 e 200
    'max_depth': [None, 10, 20, 30], # Profundidade máxima da árvore
    'min_samples_split': randint(2, 10), # Número mínimo de amostras para dividir um nó
    'min_samples_leaf': randint(1, 5) # Número mínimo de amostras em uma folha
}

random_search = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_distributions=param_dist, # Espaço de busca
    n_iter=50, # Número de combinações a testar
    cv=5, # Validação cruzada com 5 folds
    random_state=42,
    n_jobs=-1 # Paralelismo para acelerar o treinamento
)

random_search.fit(X, y)

print("Melhores parâmetros:", random_search.best_params_)
best_model = random_search.best_estimator_

Melhores parâmetros: {'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 8, 'n_estimators': 141}
```

Após essas validações, foi feito um comparativo das duas configurações para ver qual entre as duas se saiu melhor:

Imagem 26 – Código melhor combinação entre Grid Search e Random Search (parte 1):

```

7a # Configuração 1 - Hiperparâmetros fornecidos
params_1 = {'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 8, 'n_estimators': 141}
model_1 = RandomForestClassifier(**params_1, random_state=42)
model_1.fit(X_train, y_train)

# Predição
y_pred_1 = model_1.predict(X_test)

# Acurácia
accuracy_1 = accuracy_score(y_test, y_pred_1)
print(f"\nAcurácia para os parâmetros {params_1}: {accuracy_1:.2f}")

# Matriz de Confusão
cm_1 = confusion_matrix(y_test, y_pred_1)
plt.figure(figsize=(4, 3))
sns.heatmap(cm_1, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_9', 'Rock_10'], yticklabels=['Pop_9', 'Rock_10'])
plt.title('Matriz de Confusão - Parâmetros 1')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()

# Relatório de Classificação
print("\nRelatório de Classificação - Parâmetros 1:")
print(classification_report(y_test, y_pred_1))

# Recall
recall_1 = recall_score(y_test, y_pred_1, average='macro')
print(f"Recall - Parâmetros 1: {recall_1:.2f}")

```

Imagem 27 – Código melhor combinação entre Grid Search e Random Search (parte 2):

```

# Configuração 2 - Hiperparâmetros fornecidos
params_2 = {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
model_2 = RandomForestClassifier(**params_2, random_state=42)
model_2.fit(X_train, y_train)

# Predição
y_pred_2 = model_2.predict(X_test)

# Acurácia
accuracy_2 = accuracy_score(y_test, y_pred_2)
print(f"\nAcurácia para os parâmetros {params_2}: {accuracy_2:.2f}")

# Matriz de Confusão
cm_2 = confusion_matrix(y_test, y_pred_2)
plt.figure(figsize=(4, 3))
sns.heatmap(cm_2, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_9', 'Rock_10'], yticklabels=['Pop_9', 'Rock_10'])
plt.title('Matriz de Confusão - Parâmetros 2')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()

# Relatório de Classificação
print("\nRelatório de Classificação - Parâmetros 2:")
print(classification_report(y_test, y_pred_2))

# Recall
recall_2 = recall_score(y_test, y_pred_2, average='macro')
print(f"Recall - Parâmetros 2: {recall_2:.2f}")

```

Imagem 28 – Código melhor combinação entre Grid Search e Random Search (resultados parte 1):

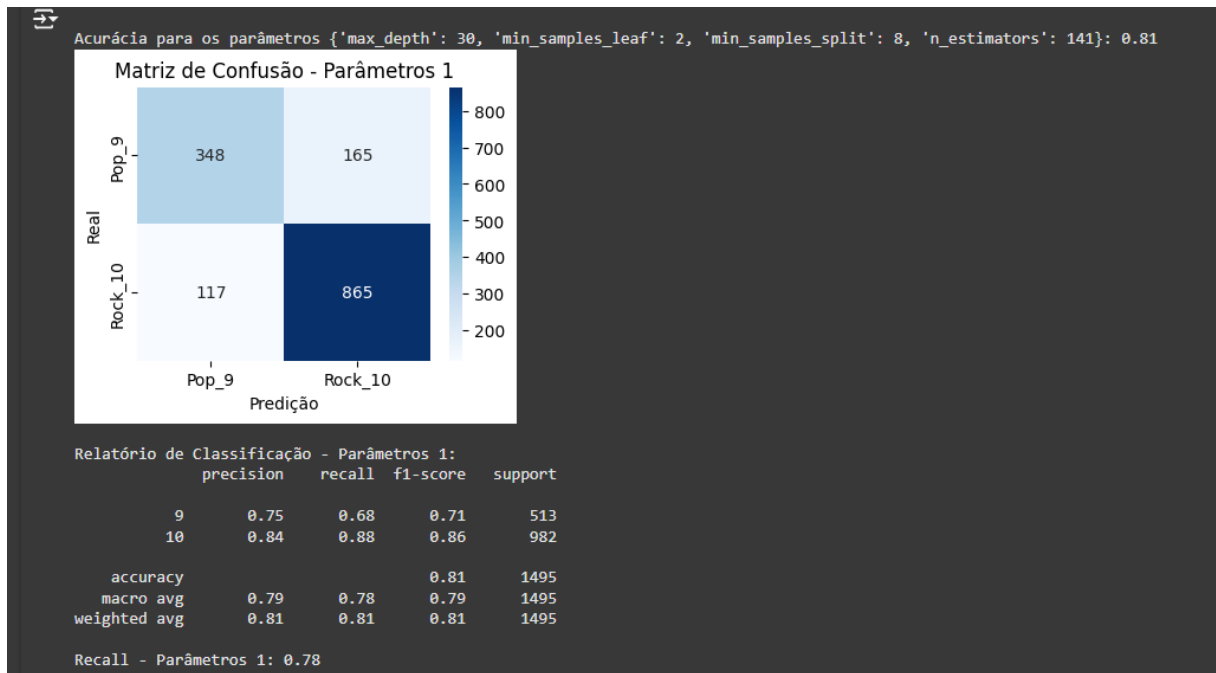
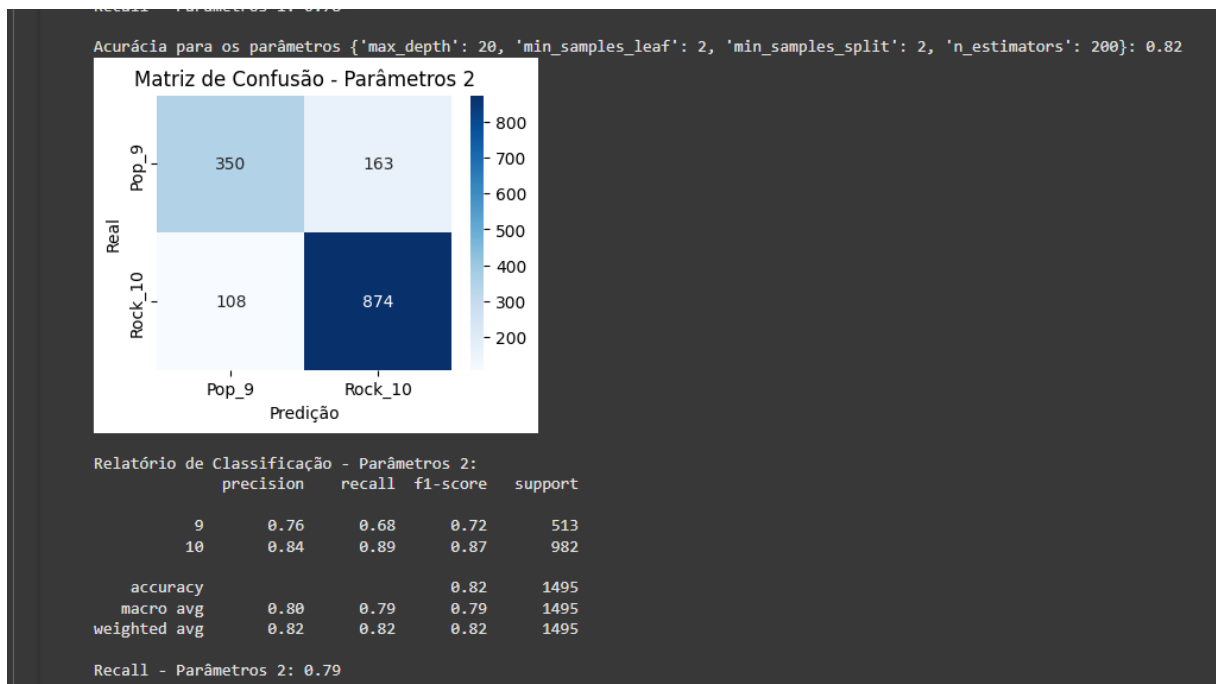


Imagem 29 – Código melhor combinação entre Grid Search e Random Search (resultados parte 2):



É possível perceber que ambos tiveram desempenho semelhante, porém os parâmetros encontrados pelo algoritmo de Grid Search foi um pouco melhor, mostrada na imagem 29.

c) Realizar feature engineering para aprimorar a qualidade dos dados de entrada:

Foi tentado algumas abordagens de mudança utilizando feature engineering, porém nenhuma teve um impacto muito grande (após alguns testes, relatório permaneceu quase que inalterado). Conforme exemplo abaixo:

Imagem 30 – Tentativa de utilizar feature engineering na coluna energy:

```
[86] train_df['energy_bin'] = pd.cut(train_df['energy'], bins=5, labels=False)
      train_df = train_df.drop(columns=['energy'])
```

6 IMPLEMENTAÇÃO DO AGENTE APRIMORADO E TREINAMENTO

6.1 Implemente as melhorias propostas e re-treine o agente com os novos ajustes

Aqui nesse ponto foi utilizado algumas técnicas diferentes para chegar em estatísticas mais altas que mostram um melhor desempenho da IA. Abaixo o que mais foi realizado e os resultados obtidos.

Imagem 31 – Trocado valor da classe para simplificar classificação:

```
train_df["Class"] = train_df["Class"].map({9: 0, 10: 1})

# Verificando se o mapeamento está correto
print(train_df["Class"].value_counts())
```

Class	count
1	4949
0	2524

Name: count, dtype: int64

Depois disso, foi feito algumas pesquisas para tentar identificar uma forma de melhorar ainda mais o modelo que estava tendo um acerto razoável, mas não tão bom. Contudo, foi encontrado uma forma de melhorar, o conceito/técnica adotada para melhorar os acertos, foi o uso de meta-modelo que nada mais é do que um modelo que descreve ou organiza outros modelos. Ele atua como um modelo sobre modelos, ajudando a compreender, estruturar ou otimizar diferentes modelos de IA.

Abaixo foi adotado utilizar alguns modelos diferentes e todos foram agrupados em um único modelo que orquestra esses outros modelos para chegar em uma solução melhor. O `StackingClassifier` é uma técnica de aprendizado de máquina que combina diferentes modelos (também chamados de estimadores base) para criar um modelo mais robusto. Cada estimador base, como Random Forest, Gradient Boosting, LightGBM, CatBoost e XGBoost, aprende de forma independente a partir dos dados de treinamento, gerando suas próprias previsões.

Essas previsões individuais são então usadas como entrada para um modelo adicional, chamado de estimador final (neste caso, a Regressão Logística). Esse modelo final combina as informações dos estimadores base para fazer a predição final, buscando capturar as vantagens de cada modelo base e reduzir erros individuais.

Em resumo, o `StackingClassifier` integra modelos diversificados para melhorar a capacidade preditiva, aproveitando a força de diferentes algoritmos em um único modelo. Ademais, os novos algoritmos adotados juntamente ao Random Forest já usado, são algoritmos de aprendizado baseados em boosting, uma técnica que combina modelos fracos (como árvores de decisão) em sequência, onde cada novo modelo corrige os erros do anterior. Eles diferem de algoritmos como Random Forest, que usa bagging (média de previsões independentes), enquanto o boosting foca no aprendizado sequencial.

Imagem 32 – Algoritmo de meta-modelo inicial:

```
#Definindo os estimadores
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100)),
    ('gb', GradientBoostingClassifier(n_estimators=100)),
    ('lgb', LGBMClassifier(n_estimators=100)),
    ('cat', CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, verbose=0)),
    ('xgb', XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=6, verbosity=0)),
]

final_estimator = LogisticRegression()

#Usando StackingClassifier com diferentes modelos
X = train_df.drop(columns=['Class'])
y = train_df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

stacking_model = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
stacking_model.fit(X, y)

#Previsões
y_pred = stacking_model.predict(X_test)

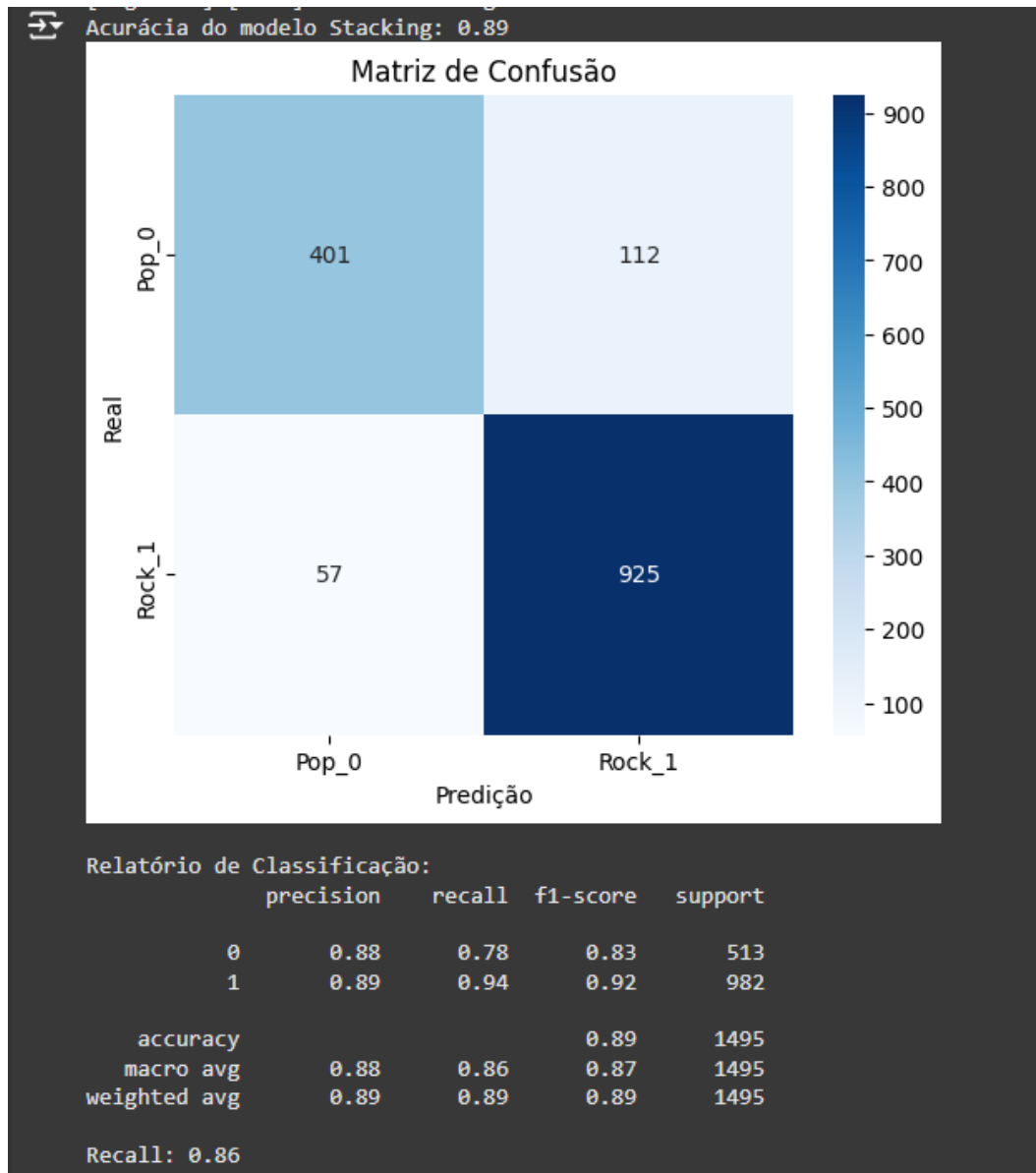
#Acurácia
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia do modelo Stacking: {accuracy:.2f}")

#Matriz de Confusão
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_0', 'Rock_1'], yticklabels=['Pop_0', 'Rock_1'])
plt.title('Matriz de Confusão')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()

#Relatório de Classificação
print("\nRelatório de Classificação:")
print(classification_report(y_test, y_pred))

#Recall
recall = recall_score(y_test, y_pred, average='macro')
print(f"Recall: {recall:.2f}")
```

Imagem 33 – Algoritmo de meta-modelo inicial (resultados):



É possível perceber nitidamente a diferença de um modelo normal para um meta-modelo na classificação de gêneros. A partir daqui os relatórios de estatísticas estão muito melhor, tem recalls e f1-score muito melhores do que eram antes, e sendo mais equilibrado entre os dois gêneros musicais. Sem falar de acurácia que passou de 82% para 89%. Mas não parou por aqui, pois hiper parâmetros também poderiam ser ajustados para cada modelo individualmente e agregar mais ao modelo principal. Abaixo foi utilizado a biblioteca SMOTE que distribui igualmente os dados dos gêneros, para que a base não fique desbalanceada, pois essa técnica de reamostragem é usada para lidar com o desequilíbrio de classes. Ela cria novas amostras sintéticas da classe minoritária em vez de simplesmente duplicar as existentes.

Imagem 34 – Algoritmo de meta-modelo de forma distribuída:

```

smote = SMOTE(random_state=42)
X = train_df.drop(columns=['Class'])
y = train_df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Definindo os estimadores
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100)),
    ('gb', GradientBoostingClassifier(n_estimators=100)),
    ('lgb', LGBMClassifier(n_estimators=100)),
    ('cat', CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, verbose=0)),
    ('xgb', XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=6, verbosity=0)),
]

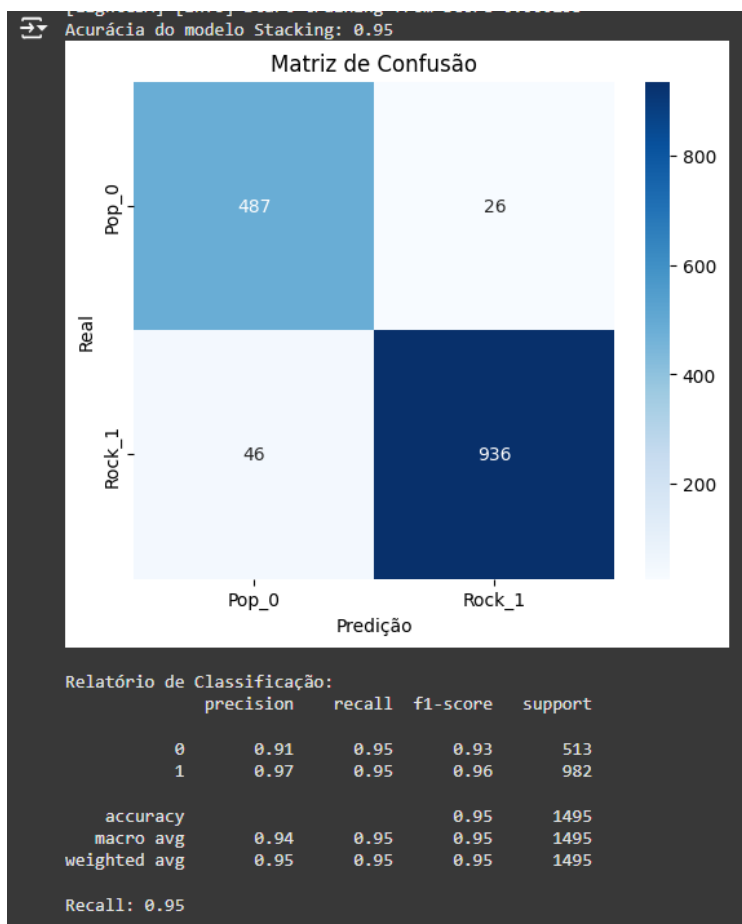
final_estimator = LogisticRegression()

# Usando StackingClassifier com diferentes modelos
stacking_model = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
stacking_model.fit(X_res, y_res)

# Previsões
y_pred = stacking_model.predict(X_test)

```

Imagem 35 – Algoritmo de meta-modelo de forma distribuída (resultados):



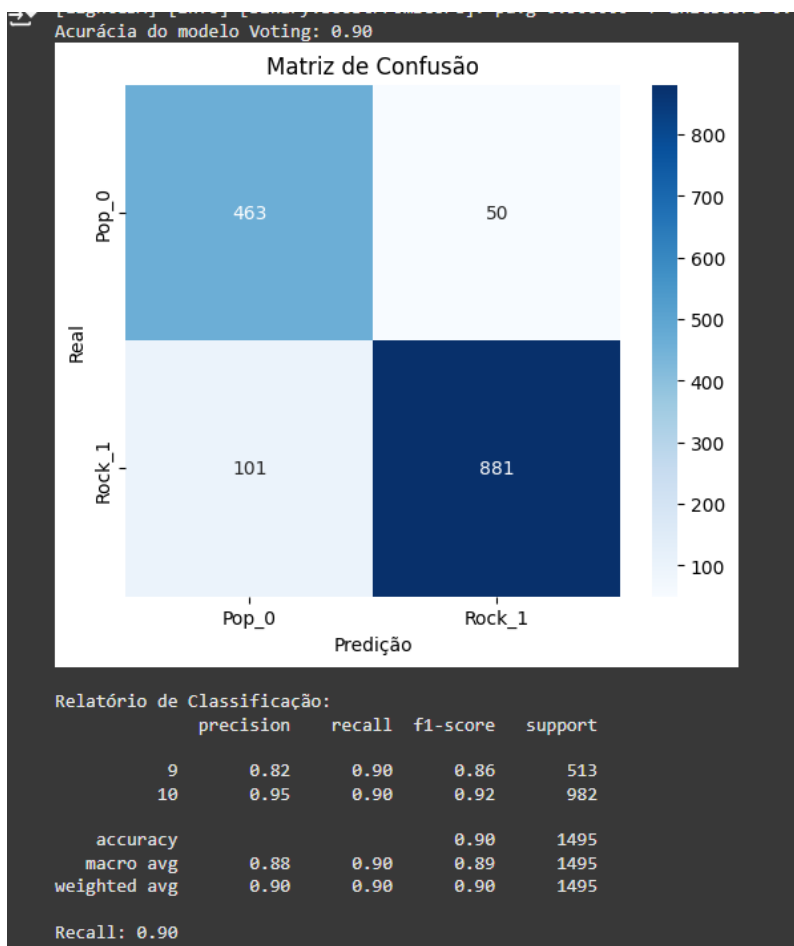
Agora o modelo está muito forte, dando estatísticas elevadas de acerto, ficando com uma acurácia de 95%, quase não havendo muitas confusões e principalmente mantendo equilíbrio entre verdadeiros positivos e verdadeiros negativos, ou seja, está balanceada. Além disso, ambos os gêneros estão sendo classificados de forma semelhante em termos de desempenho.

Para tentar melhorar ainda mais o modelo, foi tentado mudar o modelo orquestrador, para um tipo diferente de observação dos resultados dos modelos, foi usado VotingClassifier ao invés de StackingClassifier, e o resultado foi o seguinte:

Imagem 36 – Algoritmo de meta-modelo usando voting:

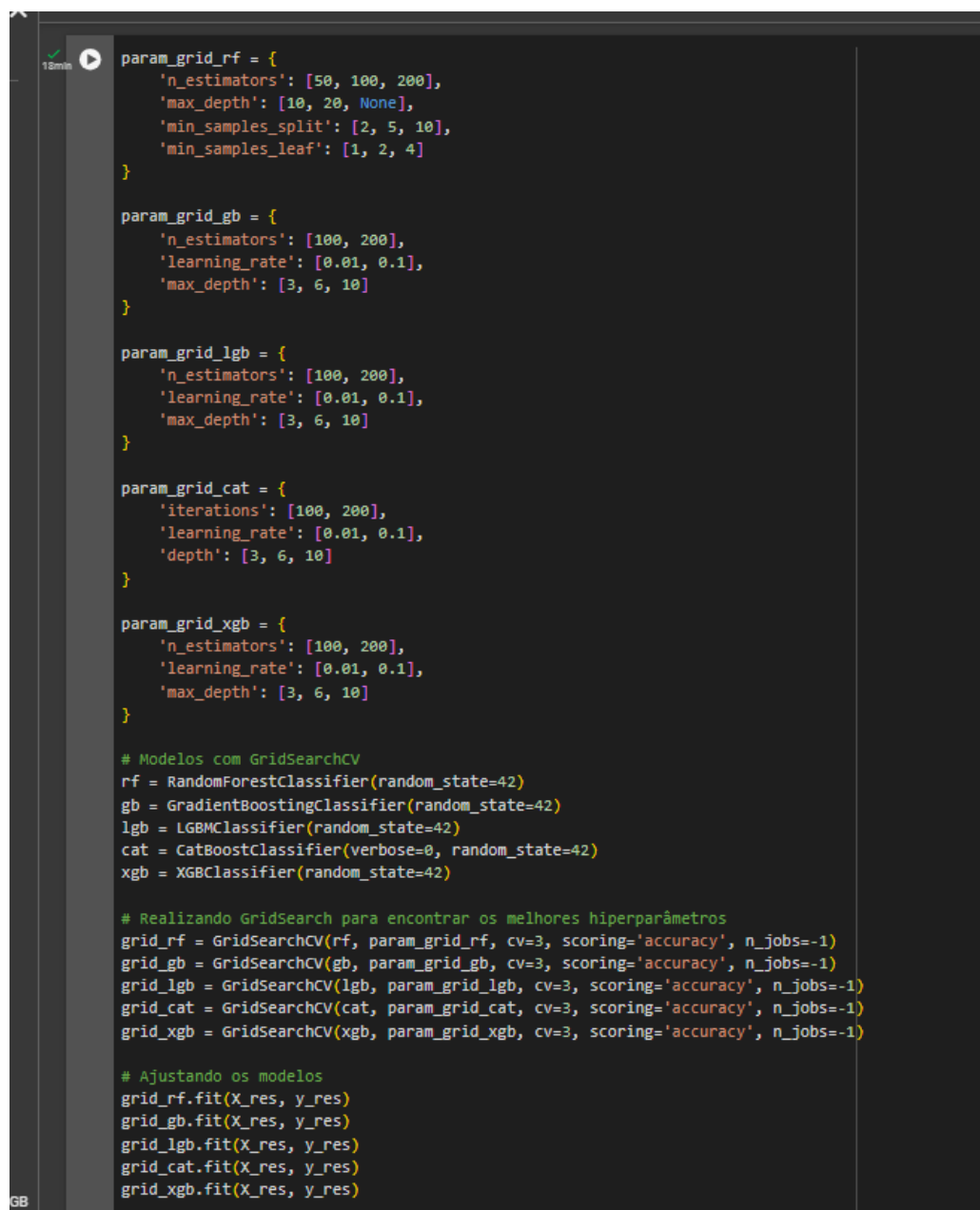
```
#Usando VotingClassifier com os modelos definidos
#Treinando com dados balanceados
voting_model = VotingClassifier(estimators=estimators, voting='soft')
voting_model.fit(X_res, y_res)
```

Imagem 37 – Algoritmo de meta-modelo usando voting (resultados):



Resultados já não foram tão bons quando utilizando StackingClassifier. Para finalizar foi buscado colocar junto outros modelos para compor esse modelo principal, porém quando adicionado KNN e Árvore binária o desempenho caia, pois o desempenho individual desses modelos não foi tão quando os outros para esse dataset e cenário específico. Por isso, foi feito somente a busca dos melhor hiper parâmetros dos modelos usados no meta-modelo com StackingClassifier que teve o melhor desempenho.

Imagem 38 – Algoritmo de meta-modelo ajustando hiper parâmetros (parte 1):



```

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_grid_gb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 6, 10]
}

param_grid_lgb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 6, 10]
}

param_grid_cat = {
    'iterations': [100, 200],
    'learning_rate': [0.01, 0.1],
    'depth': [3, 6, 10]
}

param_grid_xgb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 6, 10]
}

# Modelos com GridSearchCV
rf = RandomForestClassifier(random_state=42)
gb = GradientBoostingClassifier(random_state=42)
lgb = LGBMClassifier(random_state=42)
cat = CatBoostClassifier(verbose=0, random_state=42)
xgb = XGBClassifier(random_state=42)

# Realizando GridSearch para encontrar os melhores hiperparâmetros
grid_rf = GridSearchCV(rf, param_grid_rf, cv=3, scoring='accuracy', n_jobs=-1)
grid_gb = GridSearchCV(gb, param_grid_gb, cv=3, scoring='accuracy', n_jobs=-1)
grid_lgb = GridSearchCV(lgb, param_grid_lgb, cv=3, scoring='accuracy', n_jobs=-1)
grid_cat = GridSearchCV(cat, param_grid_cat, cv=3, scoring='accuracy', n_jobs=-1)
grid_xgb = GridSearchCV(xgb, param_grid_xgb, cv=3, scoring='accuracy', n_jobs=-1)

# Ajustando os modelos
grid_rf.fit(X_res, y_res)
grid_gb.fit(X_res, y_res)
grid_lgb.fit(X_res, y_res)
grid_cat.fit(X_res, y_res)
grid_xgb.fit(X_res, y_res)

```

Imagem 39 – Algoritmo de meta-modelo ajustando hiper parâmetros (parte 2):

```

# Mostrando os melhores parâmetros
print(f"Melhores parâmetros para RandomForest: {grid_rf.best_params}")
print(f"Melhores parâmetros para GradientBoosting: {grid_gb.best_params}")
print(f"Melhores parâmetros para LGBM: {grid_lgb.best_params}")
print(f"Melhores parâmetros para CatBoost: {grid_cat.best_params}")
print(f"Melhores parâmetros para XGBoost: {grid_xgb.best_params}")

# Definindo os estimadores com os melhores parâmetros
estimators = [
    ('rf', grid_rf.best_estimator_),
    ('gb', grid_gb.best_estimator_),
    ('lgb', grid_lgb.best_estimator_),
    ('cat', grid_cat.best_estimator_),
    ('xgb', grid_xgb.best_estimator_),
]

# Meta-modelo: Regressão Logística
final_estimator = LogisticRegression()

# Usando StackingClassifier com os melhores estimadores
# Treinando com dados balanceados
stacking_model = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
stacking_model.fit(X_res, y_res)

# Previsões
y_pred = stacking_model.predict(X_test)

# Acurácia
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia do modelo Stacking: {accuracy:.2f}")

# Matriz de Confusão
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Pop_0', 'Rock_1'], yticklabels=['Pop_0', 'Rock_1'])
plt.title('Matriz de Confusão')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()

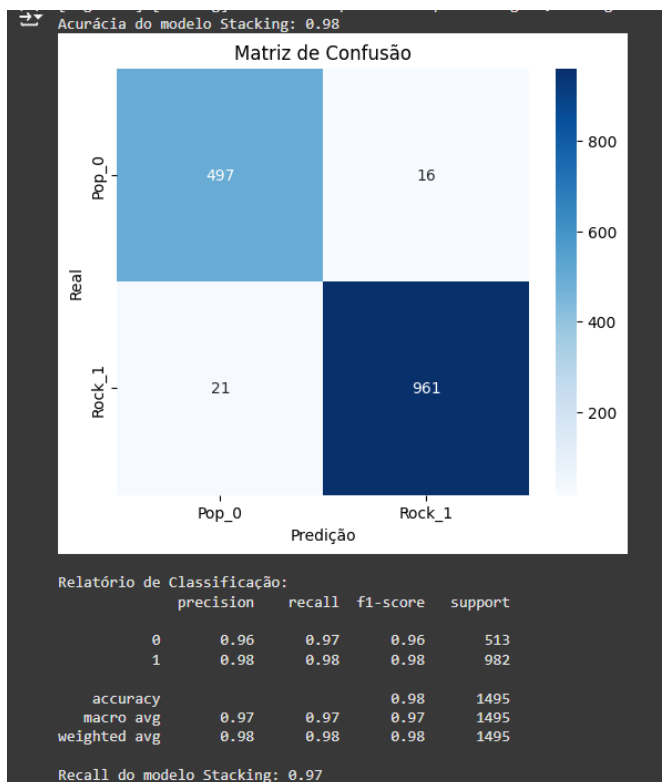
# Relatório de Classificação
print("\nRelatório de Classificação:")
print(classification_report(y_test, y_pred))

# Recall
recall = recall_score(y_test, y_pred, average='macro')
print(f"Recall do modelo Stacking: {recall:.2f}")

# Estatísticas dos Modelos Individuais
for name, model in estimators:
    y_pred_individual = model.predict(X_test)
    print(f"\nEstatísticas do modelo {name}:")
    print(f"Acurácia: {accuracy_score(y_test, y_pred_individual):.2f}")
    print(f"Recall: {recall_score(y_test, y_pred_individual, average='macro'):.2f}")
    print(f"Relatório de Classificação:\n(classification_report(y_test, y_pred_individual))")

```

Imagem 40 – Algoritmo de meta-modelo ajustando hiper parâmetros (resultados):

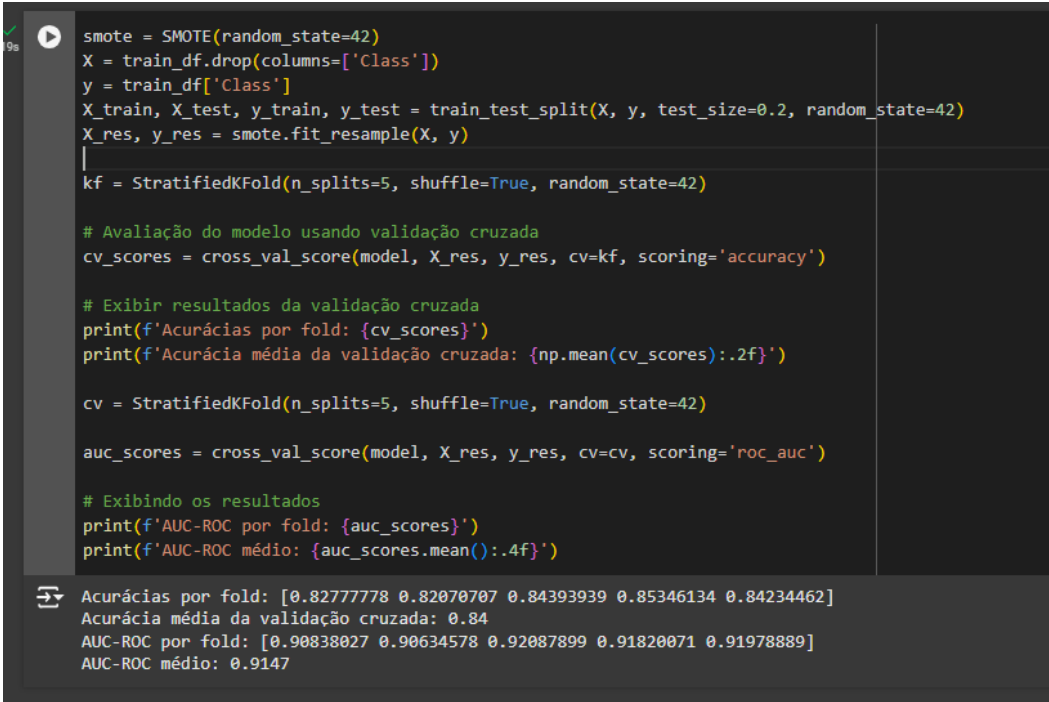


Nesse ponto foi considerado que está pronto a primeira versão do modelo classificando dois gêneros musicais, pop e rock. Com isso, foi começado a ser feito a comparação do primeiro modelo para o modelo final.

6.2 Compare os novos resultados com os do modelo base para verificar se houve melhorias significativas

Abaixo última validação de desempenho usando AUC-ROC, com validação cruzada nos testes. Foi utilizada pois é útil especialmente em cenários onde há um desbalanceamento entre as classes. Contudo, foi possível concluir que a média de acurácia de **0.84** indica um desempenho razoável, sem grandes variações entre os folds, além disso, O AUC-ROC médio de **0.9147** indica uma excelente capacidade de discriminação entre as classes. Portanto, **está bom** porque o modelo apresenta boa performance tanto na acurácia quanto na capacidade de discriminação das classes, com um bom equilíbrio entre as métricas:

Imagem 41 – Amostragem de métrica AUC-ROC:



```

smote = SMOTE(random_state=42)
X = train_df.drop(columns=['Class'])
y = train_df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_res, y_res = smote.fit_resample(X, y)

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Avaliação do modelo usando validação cruzada
cv_scores = cross_val_score(model, X_res, y_res, cv=kf, scoring='accuracy')

# Exibir resultados da validação cruzada
print(f'Acurácias por fold: {cv_scores}')
print(f'Acurácia média da validação cruzada: {np.mean(cv_scores):.2f}')

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

auc_scores = cross_val_score(model, X_res, y_res, cv=cv, scoring='roc_auc')

# Exibindo os resultados
print(f'AUC-ROC por fold: {auc_scores}')
print(f'AUC-ROC médio: {auc_scores.mean():.4f}')

```

Acurácias por fold: [0.82777778 0.82070707 0.84393939 0.85346134 0.84234462]
Acurácia média da validação cruzada: 0.84
AUC-ROC por fold: [0.90838027 0.90634578 0.92087899 0.91820071 0.91978889]
AUC-ROC médio: 0.9147

Para finalizar a comparação de desempenho é possível ver que os dados do relatório inicial que tinha uma acurácia de 82%, foi para **98%**, além as métricas como recall, precisão e f1_score darem uma boa melhorada também e de forma balanceada, principalmente no recall

do pop, que estava bem baixo e acabou terminando com **97%** de acertos. Então após as mudanças o modelo melhorou muito.

7 VALIDAÇÃO E ANÁLISE FINAL

7.1 Avalie o desempenho do agente aprimorado com o conjunto de dados de teste

Isso foi feito ao decorrer do projeto, porém foi feito também para o último cenário também, que está com o resultado e conclusão das estatísticas no tópico 6.

7.2 Documente a performance final e avalie se o agente atende ao objetivo inicial

A performance final foi muito boa, conforme as imagens do último teste, teve poucas confusões, com acurácia, f1_score, recall e precisão muito altos em todo os dois gêneros musicais. Fiz um teste extra tentando treinar com o melhor modelo todos os gêneros musicais, e tive até um bom retorno, para uma possível nova versão já daria de ir aprimorando para chegar perto da melhor estatística que hoje é só dois gêneros. Porém, não irei detalhar muito pois não é o foco desse escopo.

7.3 Crie um conjunto de testes para demonstrar o funcionamento do agente em diferentes cenários

Abaixo o conjunto de teste usados, pegos também no Kaggle, só que de outro arquivo do dataset, que vem junto com o arquivo de treino, usado para demonstrar o funcionamento do modelo de classificação pronto (os demais testes já usaram dados de teste em massa, aqui é mais demonstração do funcionamento):

Imagem 42 – 2 músicas pop:

```
#Pop
["MC Hammer", "U Can't Touch This", 0.867, 0.517, 0, 0.0875, 0.00456, 0.000339, 0.866, 133.148, 4],
["R3HAB", "Whiplash - Acoustic", 0.556, 0.188, 1, 0.0316, 0.494, 0, 0.184, 138091, 4],
```

Imagem 43 – 2 músicas rock:

```
#Rock
["Sharon Van Etten", "Comeback Kid", 0.543, 0.927, 1, 0.0755, 0.000342, 0.543, 0.354, 120097, 4],
["Jimmy Eat World", "All The Way (Stay)", 0.576, 0.931, 1, 0.0563, 0.00085, 0, 0.735, 133986, 4],
```

Resultados:

Imagem 44 – Resultado da demonstração:

```
# Conjunto de dados fornecido
data = [
    #Pop
    ["MC Hammer", "U Can't Touch This", 0.867, 0.517, 0, 0.0875, 0.00456, 0.000339, 0.866, 133.148, 4],
    ["R3HAB", "Whiplash - Acoustic", 0.556, 0.188, 1, 0.0316, 0.494, 0, 0.184, 138091, 4],
    #Rock
    ["Sharon Van Etten", "Comeback Kid", 0.543, 0.927, 1, 0.0755, 0.000342, 0.543, 0.354, 120097, 4],
    ["Jimmy Eat World", "All The Way (Stay)", 0.576, 0.931, 1, 0.0563, 0.00085, 0, 0.735, 133986, 4],
]

# Colunas do conjunto de dados
columns = ["Artist Name", "Track Name", "danceability", "energy", "mode", "speechiness", "acousticness", "instr

# Criando o DataFrame
testeGeneros = pd.DataFrame(data, columns=columns)

label_encoder_artist = LabelEncoder()
label_encoder_track = LabelEncoder()

testeGeneros["Artist Name"] = label_encoder_artist.fit_transform(testeGeneros["Artist Name"])
testeGeneros["Track Name"] = label_encoder_track.fit_transform(testeGeneros["Track Name"])

# Previsão usando o modelo
y_pred = stacking_model.predict(testeGeneros)

# Exibindo os resultados
print(y_pred)
```

[0 0 1 1]

7 CONCLUSÃO

Foi possível concluir que o modelo iniciou de forma muito ruim com todos os gêneros, e após filtragem e pré-processamento das variáveis, mantendo somente o que era pertinente e causavam impacto, o modelo começou a ficar mediano, principalmente quando filtrei dois gêneros para começar. Após todo o processo de buscar o melhor modelo foi visto que para dar uma melhoria significativa o modelo precisaria se tornar um meta-modelo, juntando vários modelos diferentes com precisões distintas, até chegar em uma acurácia de 98%, com pouquíssimas confusões entre os dois gêneros, mantendo o equilíbrio entre verdadeiros positivos e verdadeiros negativos, por fim, na validação cruzada manteve uma boa métrica com o modelo final e poderia até evoluir o modelo para classificar todos os gêneros, claro que perderia um pouco de performance inicialmente, mas a medida que novos dados fossem usados para treino esse modelo poderia ir evoluindo, pois já está com um desempenho regular para começar.

8 REFERÊNCIAS

AMARAL, Fernando. Formação Completa Inteligência Artificial e Machine Learning.
Disponível em: <https://www.udemy.com/course/inteligencia-artificial-e-machine-learning/>.
Acesso em: 15 nov. 2024.