Resultados – Teste Prático da Smarkio

Nome do candidato: Rafael Greca Vieira
Nome da vaga: Data Science e Machine Learning

ATIVIDADES

1- Análise exploratória dos dados utilizando estatística descritiva e inferencial, considerando uma, duas e/ou mais variáveis.

O problema em questão se trata de um problema de classificação. Isso quer dizer que deverá ser atribuída uma uma classe, dentro de todas as possibilidades possíveis, para uma determinada entrada. No problema em questão devemos prever qual classe, representada por um número inteiro, se encaixa um determinado dado.

Possuímos um conjunto com 643 dados (também podemos chamar de linha) e 4 colunas. As colunas são: *pred_class*, classe que foi indetificada pelo modelo; *Probabilidade*, a probabilidade da classe que o modelo identificou; *Status*, status da classificação de acordo com um especialista; *True_class*, é a verdadeira classe. Se o valor de *pred_class* for igual ao do *true_class* quer dizer que o modelo acertou.

É importante ressaltar que os dados que serão utilizados para análise são os que possuem status **approved.** Pois os mesmos já foram avaliados e aprovados por um especialista, então é mais confiável utilizá-los.

1.1- Examinando as colunas

```
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Pred_class     643 non-null    int64
 1   probabilidade  643 non-null    float64
 2   status         643 non-null    object
 3   True_class     181 non-null    float64
dtypes: float64(2), int64(1), object(1)
```

Como podemos ver na figura a cima, duas colunas são do tipo **int**, uma do tipo **float** e a outra do tipo **object.** Essa coluna float, *true_class*, pode ser tratada como uma coluna **int**, já que não possui valores com casas decimais. Podemos separar as colunas em duas categorias: numérica e categórica.

A numérica é composta por colunas do tipo **int** e **float**. Sendo assim, *pred_class*, *probabilidade* e *true_class* fazem parte dessa categoria. Já a categórica é composta por colunas do tipo **object.** Então *status* faz parte dessa categoria.

Somente uma coluna possui valores nulos: a *true_class*. Esses valores nulos serão substituídos pelo valor do *pred_class*.

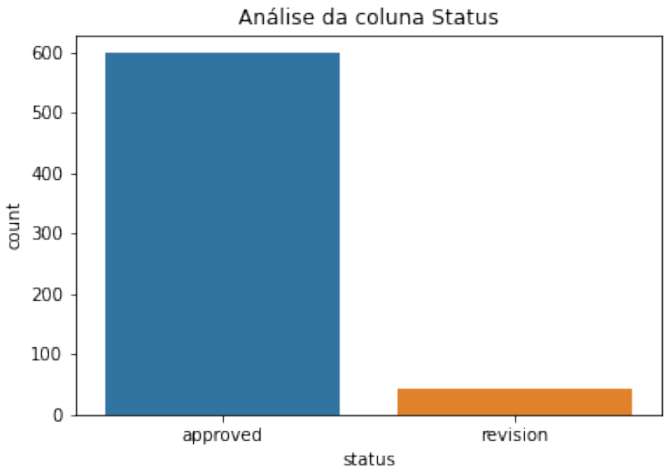
1.2- Informações estatísticas das colunas

	Pred_class	probabilidade	status	True_class
count	643.000000	643.000000	643	181.000000
unique	NaN	NaN	2	NaN
top	NaN	NaN	approved	NaN
freq	NaN	NaN	600	NaN
mean	52.712286	0.622436	NaN	38.574586
std	37.602068	0.266811	NaN	39.581017
min	2.000000	0.043858	NaN	0.000000
25%	12.000000	0.408017	NaN	0.000000
50%	59.000000	0.616809	NaN	24.000000
75%	81.000000	0.870083	NaN	74.000000
max	118.000000	1.000000	NaN	117.000000

As variáveis **unique, top e freq** significam, respectivamente, **valores únicos, valor com maior frequência** e **a quantidade de vezes o valor com maior frequência aparece.** Essas variáveis são estão disponíveis para colunas categóricas. As variáveis **min** e **max** mostram, respectivamente, o valor mínimo e máximo das colunas numéricas. As variáveis **mean** e **std** mostram o valor da média e do desvio padrão das colunas numéricas. As variáveis **25%, 50%** e **75%** mostrando o valor do quartil inferior, a mediana e o quartil superior, respectivamente, das colunas numéricas.

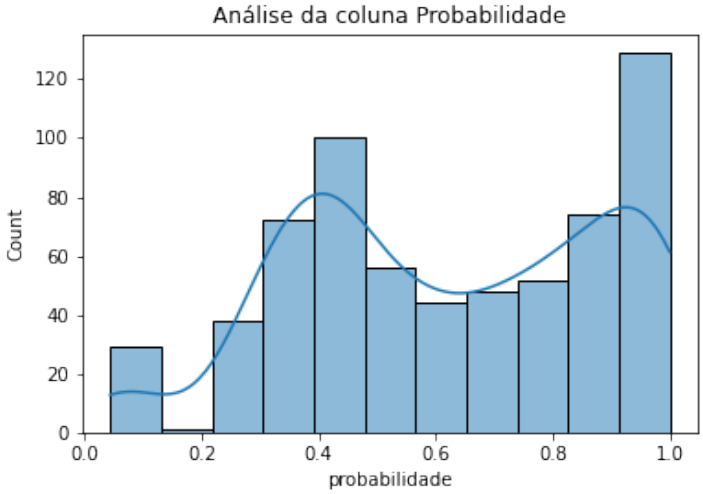
1.3- Análise da coluna Status

Como vimos anteriormente, a coluna **status** possui dois valores únicos. Agora vamos ver quais são eles e quantas vezes aparecem na coluna.



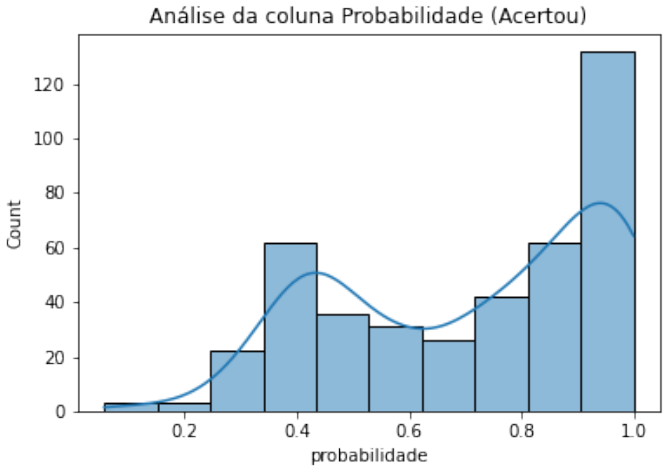
Através do gráfico a cima podemos concluir que os dois possíveis valores são: **approved** e **revision.** Approved aparece 600 vezes, aproximadamente 93% do total, e revision aparece 43 vezes, aproximadamente 7% do total. Então podemos concluir que existem mais dados que foram aprovados pelo especialista do que dados que precisam de uma revisão.

1.4- Análise da coluna Probabilidade



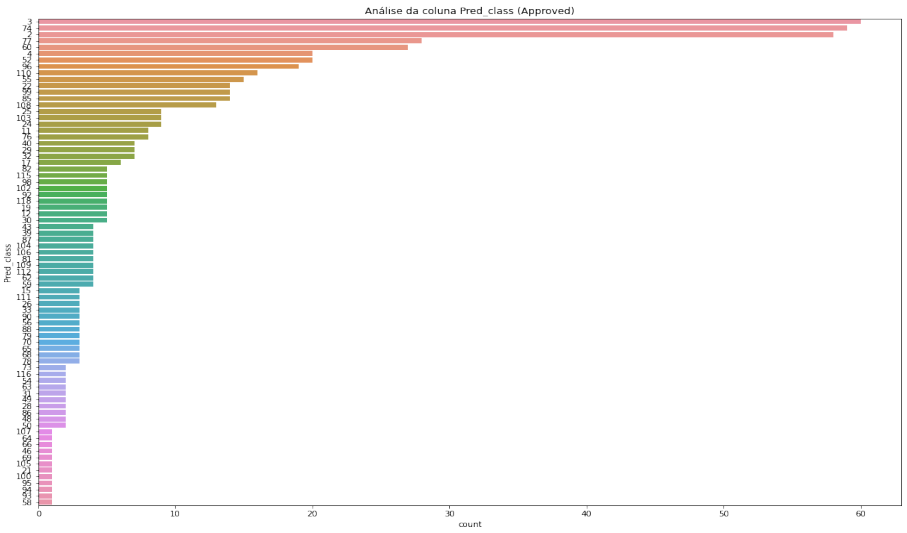
Através do gráfico do histograma a cima podemos perceber que quase representa uma distribuição normal, porém o lado direito do gráfico acaba prejudicando. Além disso, é possível visualizar que o modelo possui dois picos: entre os valores 0.8 e 1.0 e 0.3 e 0.5. Isso quer dizer que ao mesmo tempo que o modelo tem bastante certeza da classe que foi prevista está certa (em uma grande parte das vezes), em outras vezes ele não tem tanta certeza assim. O que acaba deixando a média da probabilidade em torno de 50%.

Na figura abaixo vemos que, quando o modelo acerta a classe, o pico entre os valores 0.8 e 1.0 fica maior e 0.3 e 0.5 menor. Ou seja, quanto maior a probabilidade, maior a chance de acertar.



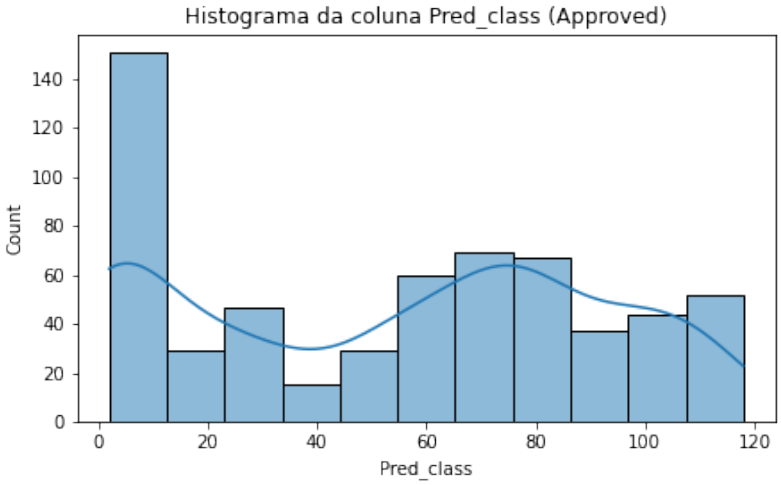
1.5- Análise da coluna Pred_class

A coluna *Pred_class* será analisada para que possamos verificar quais classes aparecem com maior frequência e sua distribuição. Podemos ver na figura abaixo que não é distribuída uniformemente. Ou seja, aparecem com frequências diferentes, exemplo: as classes que mais aparecem são 2, 3, 4, 52, 60, 74, 77 e 96. A classe *Pred_class* possui 80 valores diferentes para classe, 76 desses aparecem no dados que possuim status **approved.** Como a coluna *True_class* apresenta 73 classes, isso quer dizer que foi previsto classes que não aparecerem na *True_class.*



Obs: a imagem com uma qualidade melhor pode ser encontrada dentro do código ou na pasta de imagens.

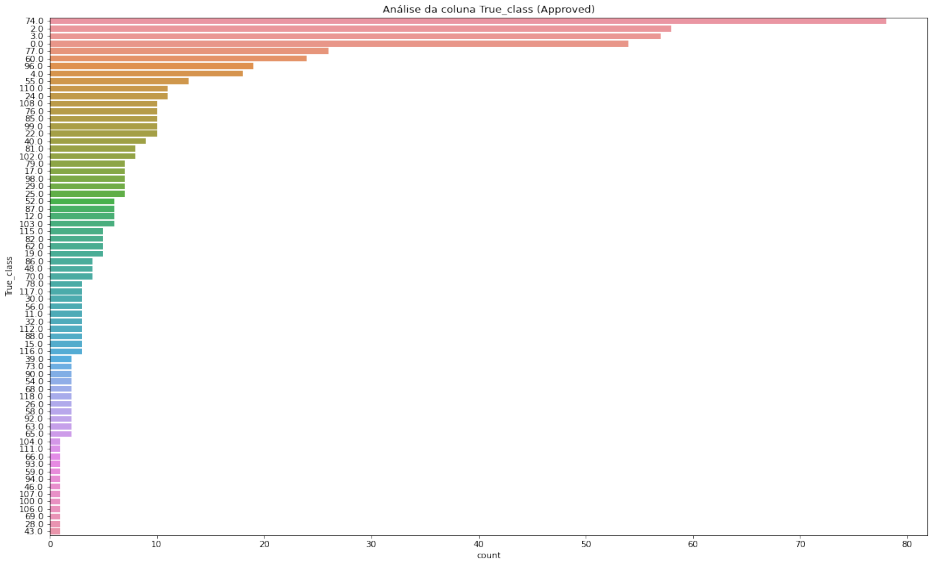
Veremos, na análise da coluna True_class, que esse comportamento era esperado devido a distribuição das classes não ser normal. Isso quer dizer que as classes que possuem uma frequência maior na *True_class* também aparecerão com grande frequência na coluna *Pred_class.* Também podemos perceber no histograma abaixo que o comportamento entre as duas colunas são semelhantes.



Ela também pode ser considerada assimétrica positiva. Estatisticamente falando, a moda (entre 2 e 10), a média (52,71) e a mediana (59). É interessante perceber também que não foi prevista nenhuma classe 0, mesmo ela tendo aparecido na coluna *True_class.* Podemos induzir duas coisas: 1- o modelo tem dificuldades para reconhecer essa classe; 2- a classe 0 não existe e pode ter sido um erro de digitação ou até mesmo um outlier.

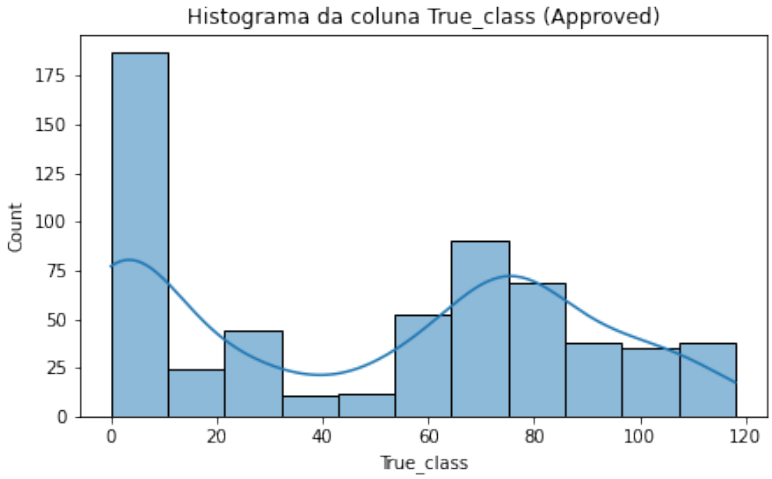
1.6- Análise da coluna True_class

A coluna *True_class* será analisada para que possamos verificar como ocorre a distribuição das classes. Podemos ver na figura abaixo que não é distribuída uniformemente. Ou seja, aparecem com frequências diferentes, exemplo: as classes que mais aparecem são 0, 2, 3, 4, 60, 74, 77 e 96. A classe *True_class* possui 73 valores diferentes para classe, 69 desses aparecem no dados que possuim status **approved.**



Obs: a imagem com uma qualidade melhor pode ser encontrada dentro do código ou na pasta de imagens.

O mesmo pode ser visualizado através do histograma da coluna, que pode ser visualizado no gráfico abaixo.

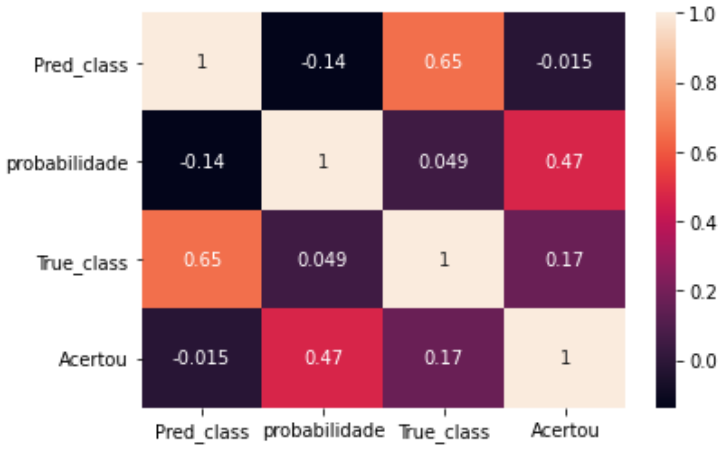


Podemos perceber também que o comportamento se assemelha um pouco a de um dado assimétrico positivo. Estatisticamente falando, a moda (entre 0 e 10) está à esquerda da média (38.57) e da mediana (24). O que comprova a ideia.

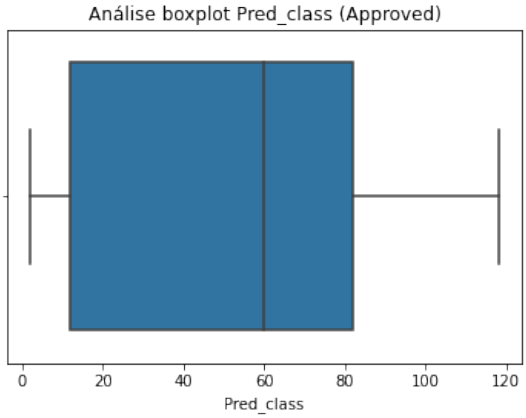
Isso acaba deixando o modelo mais tendencioso, pois estará mais acostumado com as classes que aparecem com uma frequência maior e, consequentemente, irá ter uma chance maior de acertar. Já pelo outro lado, quando é uma classe que apareceu com menos frequência a chance é menor.

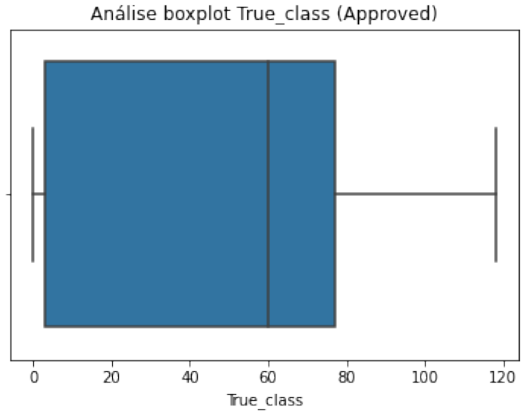
1.7- Outras análises

Como mostra a figura abaixo, a coluna **probabilidade** tem uma correlação relativamente alta com a coluna **acertou** (que é de um tipo booleano para mostrar se o modelo acertou ou não a classe).



Tanto a classe **True_class** e **Pred_class** não possuem outliers. Como podemos ver nas imagens abaixo:





As classes que tiveram maior erro no modelo foram: 0, 74, 2, 3, 81 e 24. Já as classes que tiveram o maior acerto no modelo foram: 74, 3, 2, 60, 77 e 96. As imagens dos gráficos não serão disponibilizadas já que são muito grandes e não ficaria legível. Elas podem ser acessadas na pasta **images** do projeto ou no código Python (**main.ipynb)**.

2- Calcule o desempenho do modelo de classificação utilizando pelo menos três métricas.

As métricas utilizadas para a avaliação do modelo foram: **Confusion Matrix, Accuracy Score, F1 Score, Recall Score** e **Precision Score.** As funções utilizadas que realizam as métricas estão disponíveis na biblioteca *Scikit Learn.*

2.1- Confusion Matrix

A imagem da confusion matrix (matriz de confusão) pode ser encontrado no arquivo Python (**main.ipynb)**. Ela foi utilizada para dar uma visão de quais classes o modelo mais se confunde. Quanto mais escura for a área, maior é a ocorrência. Com isso concluímos que o modelo confunde bastante a classe 0 com a 28 e a 44 com a 1.

2.2- Accuracy Score

Será explicado na atividade 4.

Score obtido = 0.6983333333333334

2.3- F1 Score

Utiliza o valor do Recall Score e do Precision Score. Foi utilizada o valor **weighted** para **average** já que as classes não são uniformemente distribuídas e é a que retorna o melhor score.

Score obtido = 0.6831017455927221

2.4- Recall Score

Será explicado na atividade 4. Foi utilizada o valor **weighted** para **average** já que as classes não são uniformemente distribuídas e é a que retorna o melhor score.

Score obtido = 0.6983333333333334

2.5- Precision Score

Será explicado na atividade 4. Foi utilizada o valor **weighted** para **average** já que as classes não são uniformemente distribuídas e é a que retorna o melhor score.

Score obtido = 0.6988175116236557

3- Crie um classificador que tenha como output se os dados com status igual a revision estão corretos ou não (Sugestão: Técnica de cross-validation K-fold).

Para a atividade em questão foi criado um classificador usando o Random Forest Classifier. Os dados utilizados para treinar o classificador foram os que tinham o status como **approved.** Isso foi feito porque que esses dados foram aprovados por um especialista, então seria mais confiável utilizá-los no treinamento. Antes disso eles foram devidamente tratados, onde todos os seus valores nulos foram substituídos pelo valor do seu **Pred_class (classe prevista pelo modelo).**

Por se tratar de um classificador simples, o **score** obtido foi de, aproximadamente, **0.814.** Isso poderia ser melhorado utilizando um classificador mais eficiente, como o XGBoost ou LightGBM, ou utilizando técnicas para melhorar os hiper parâmetros do modelo, como a função GridSearchCV, RandomSearchCV da biblioteca *Scikit Learn*, ou a biblioteca Optuna. O código do classificador pode ser encontrado no arquivo Python (**main.ipynb)**.

4- Compare três métricas de avaliação aplicadas ao modelo e descreve sua diferença.

As métricas utilizadas para a avaliação do modelo criado na atividade anterior foram: **Accuracy Score, Recall Score** e **Precision Score.** As funções utilizadas que realizam as métricas estão disponíveis na biblioteca *Scikit Learn.*

Antes de explicarmos as métricas é necessário comentar sobre os valores **verdadeiro positivo, verdadeiro negativo, falso positivo** e **falso negativo**, pois todas as métricas utilizam pelo menos dois desses valores. Será usado duas classes fictícias (X e Y) para facilitar a explicação.

Verdadeiro positivo (VP): é quando o modelo/classificador identifica a classe como X e ele realmente pertence a essa classe.

Falso positivo (FP): é quando o modelo/classificador identifica que é a classe X, mas ela pertence a classe Y.

Falso negativo (FN): é quando o modelo/classificador identifica que é a classe Y, mas ela pertence a classe X.

Verdadeiro negativo (VN): é quando o modelo/classificador identifica que é a classe Y e ele realmente pertence a essa classe.

4.1- Accuracy Score

Essa métrica utiliza os valores do verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo para calcular a precisão do modelo. Ela utiliza a seguinte fórmula:

Score = (VP + VN) / (VP + VN + FP + FN)
Score obtido = 0.813953488372093

4.2- Recall Score

Essa métrica utiliza os valores do verdadeiro positivo e falso negativo para calcular a precisão do modelo. Ela utiliza a seguinte fórmula:

Score = VP / (VP + FN)
Score obtido (Weighted) = 0.813953488372093
Score obtido (Macro) = 0.5555555555555556

Além disso também foi utilizada uma variável a mais da métrica: a **average.** Quando é utilizado o valor **macro**, quer dizer que será calculado um score para cada classe existente e tirar a média delas. Quando é utilizado o valor **weighted**, quer dizer que será calculado um score para cada classe existente e cada um terá um peso diferente de acordo com a quantidade de instâncias verdadeiras da sua classe.

4.3- Precision Score

Essa métrica utiliza os valores do verdadeiro positivo e falso positivo para calcular a precisão do modelo. Ela utiliza a seguinte fórmula:

Score = VP / (VP + FP)
Score obtido (Weighted) = 0.813953488372093

Score obtido (Macro) = 0.5555555555555556

Também possui a variável **average,** igual o Recall Score, e também foram utilizados os mesmos valores.

5- Crie um classificador, a partir da segunda aba – NLP do arquivo de dados, que permita identificar qual trecho de música corresponde às respectivas artistas listadas (Sugestão: Naive Bayes Classifier).

Para a atividade em questão foi criado um classificador usando o Naives Bayes Classifier. Antes dele ser criado foi necessário realizar o tratamento das variáveis. No exercício em questão só utilizamos a coluna **Letra**, já que a coluna **Artista** é o nosso objetivo a ser encontrado.

Para o tratamento da variável foi utilizado o método *Bags of Words,* que é feito utilizando a função *CountVectorizer* da biblioteca *Scikit Learn*. Nela será feito a tokenização das palavras, onde será associado um número inteiro para cada uma dela, e depois irá armazenar quantas vezes cada palavra apareceu no trecho da música. Além disso essa função também realizará o pré processamento e a remoção de *stop words* (palavras que não agregam valor).

Por se tratar de um classificador simples, o **score** obtido foi de **0.75.** Isso poderia ser melhorado utilizando um classificador mais eficiente, como o SVM, ou utilizando técnicas para melhorar os hiper parâmetros do modelo, como a função GridSearchCV, RandomSearchCV da biblioteca *Scikit Learn,* ou a biblioteca Optuna. Além disso, pode-se utilizar a biblioteca *NLTK*, que foi desenvolvida para tratar problemas de NLP. O código do classificador pode ser encontrado no arquivo Python (**main.ipynb)**.