Trabalho Final de Fundamentos de Sistemas Inteligentes - Classificação de Subgêneros da Música Eletrônica utilizando Machine Learning

Arthur da Veiga Feitoza Borges - 13/0050725

***Abstract*— Este trabalho é um estudo referente a classificação de subgêneros musicais dentro do gênero "EDM - Electronic Dance Music" utilizando Support Vector Machines e Modelo de Misturas Gaussiano.**

I. Introdução

Desde a ascensão da Música Eletrônica na década de 60, temos visto uma crescente diversidade dentro do gênero. Hoje em dia há mais de 60 subgêneros, e vários deles são semelhantes uns aos outros, seja em aspectos instrumentais, rítmicos, temporais, harmônicos, melódicos ou até mesmo como o ganho é organizado ao longo do espectrograma de cada música, isto é, como cada música é mixada de acordo com seu subgênero.

No artigo "Music Genre Classification via Machine Learning", de Zhiwei Gu, Li Guo e Tianchi Liu [3], vemos um estudo relacionado à recuperação de informações musicais (Music Information Retrieval - MIR) utilizando Machine Learning para 16 gêneros musicais diferentes, entre eles o Rock, Folk, Hip-hop, Eletrônica e até o Instrumental. Mas, neste artigo foi abordado apenas a nível de gêneros. Portanto, vemos um problema relevante a ser estudado que é o seguinte: *é possível classificar uma música eletrônica a nível de subgênero de forma eficiente utilizando Machine Learning?*

Esse problema se reforça uma vez que Música Eletrônica em geral sempre remeteu à inovação, desde o seu marco inicial, no final da década de 40 com o francês Pierre Schaeffer, que criou o musique concrète, que utiliza de toca-discos para gerar ruídos que variam devido à manipulação sonora que se fazia em sua composição. E esta inovação é bem forte dentro do escopo da Música Eletrônica a ponto de descartar a ideia de subgênero e gênero musical, já que há vários produtores que utilizam de outras características de gêneros e subgêneros como base para suas músicas. Por causa disso, a definição do próprio subgênero para cada música pode ser uma questão.

II. Modelo

O problema, como definimos na introdução, é de classificar subgêneros de Música Eletrônica utilizando-se de classificadores, para definir qual o subgênero musical correspondente à música dada com alta acurácia e precisão.

Mas, para resolvermos este problema, é extremamente necessário fazer um pré-processamento minucioso dos dados. Estes dados são vários arquivos .mp3 inseridos em pastas, cujos nomes são correspondentes aos subgêneros musicais que serão as classes.

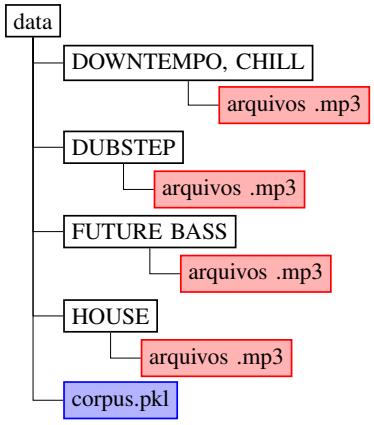


Fig. 1. Organização de diretório dos dados que serão usados para a classificação.

O dataset é composto por 323 arquivos .mp3, os quais:

- 70 arquivos estão na pasta "DOWNTEMPO, CHILLOUT"
- 24 arquivos estão na pasta "DUBSTEP"
- 55 arquivos estão na pasta "FUTURE BASS"
- 174 arquivos estão na pasta "HOUSE"

O corpus.pkl é o objeto resultante das extrações das features. Como fazer a extração demanda muito tempo para uma maior quantidade de músicas, o objeto é salvo para evitar o reprocessamento dos mesmos dados. Dito isso, detalharemos agora sobre os elementos utilizados para tentar solucionar os problemas.

A. *Bibliotecas utilizadas*

A linguagem de programação adotada foi o Python3, por ser muito poderoso e ter uma vasta gama de bibliotecas à nossa disposição. As seguintes bibliotecas foram adotadas para este trabalho;

- **`sklearn`**: Biblioteca para o pré-procesamento do corpus (normalização e redução de dimensionalidade das features), e para classificação do corpus devidamente organizado (com SVM e GMM, além da GridSearchCV, para validação cruzada).
- **`os`**: Biblioteca para manipulação de caminhos de diretório, para pegar `os` dados corretamente.
- **`numpy`**: Biblioteca para manipulação das features. Usado em todo o código.
- **`matplotlib`**: Biblioteca para plotting dos resultados de cada processo.
- **`libROSA`**: Biblioteca para análise e captura de features de áudio.
- **`pickle`**: Biblioteca para salvar/carregar corpus.

Todas as bibliotecas estão devidamente especificadas na seção "IMPORTS" do código *ml_subgenre_classification_code*, disponibilizado em `https://github.com/arthurveiga/ml_edm_subgenre_classification`.

B. *Features: quais features vamos utilizar?*

Para este trabalho consideramos os seguintes aspectos para adquirir as features a partir das músicas:

- Features Temporais: Tempo, Taxa de Cruzamento pelo Zero
- Features Espectrais: Porcentagem de Baixa energia, Coeficientes de Mel-frequências Cepstrais (MFCCs), Rolloff, Largura de Banda, Contraste e Centróide Espectrais.

1) ***Porcentagem de Baixa Energia - low_energy_percentage:*** A energia de um sinal é referente à magnitude total do sinal. Para sinais de áudio, é praticamente o quão alto o sinal é. A energia como sinal é definida da seguinte forma:

$$\sum_n |x(n)|^2$$

Já a energia RMS em um sinal é definido como:

$$\sqrt{\frac{1}{N}\sum_n |x(n)|^2}$$

Assim, a Baixa Energia (Low Energy) é a porcentagem do audio que tem menos energia RMS que o normal da totalidade da forma de onda. Como cada amostra é dividida em frames, tendo propósito de computação, podemos usar esta feature como uma medida efetiva de distribuição de amplitude. Há músicas que tem uma distribuição amplitudinal mais concentrada nos graves, como o Dubstep. Já há músicas que tem uma distribuição amplitudinal um pouco mais focado nos médios-agudos, como no Future Bass.

2) ***Taxa de Cruzamento pelo Zero - zero_crossing_rate:*** É o número de vezes em que o sinal cruza o eixo horizontal. Pode ser usado para medir a ruidosidade de sinais de áudio. O zero_crossing_rate pode ser maior para músicas com timbres mais distorcidos.

3) ***Tempo:*** É a velocidade na qual uma música é tocada. Sua unidade de medida é *batidas por minuto* (BPM). Esta feature é essencial no escopo da Música Eletrônica, pois é uma das principais características para classificar uma música.

- Chillout, Downtempo varia entre 80 e 115 BPM.
- Dubstep varia entre 70 e 90 BPM (140 a 180 BPM). Em Dubstep geralmente existem batidas duplas, dando a impressão de tempo duplicado (70 para 140 BPM).
- House varia entre 120 a 130 BPM. É um dos subgêneros mais tradicionais do escopo da Música Eletrônica.
- Já Future Bass não tem exatamente uma predefinição de tempo, já que seu alcance de tempo é bem grande (80 a 170 BPM). O que define o Future Bass são os instrumentos utilizados, que têm timbres únicos, além de suas progressões rítmicas, harmônicas e melódicas singulares.

4) ***Rolloff Espectral - spectral_rolloff:*** É a frequência na qual, logo abaixo dela, 85% da distribuição amplitudinal está. É usado como uma medida da forma espectral do sinal.

5) ***Coeficientes de Mel-frequências Cepstrais (MFCCs):*** São feitos para descrever sucintamente um espectro suavizado de um sinal. Eles sumarizam de forma compacta uma forma modificada e suavizada da envoltória espectral do sinal. Geralmente pedem-se 13 MFCCs para aplicações de reconhecimento de voz, mas, segundo Tzanetakis e Cook [7], 5 já bastam para a máxima efetividade na música.

6) ***Largura de Banda Espectral - spectral_bw:*** É o intervalo de onda no qual a quantidade espectral irradiada não é menos que a metade do valor máximo.

7) ***Contraste Espectral - spectral_contrast:*** O contraste espectral é dividido em 7 sub-bandas. Ele consid-

era o pico espectral, o vale espectral e suas diferenças em cada sub-banda de frequência. É uma feature muito útil para análise amplitudinal entre músicas.

*8) **Centróide Espectral - spectral_centroid**:* Por fim, o Centróide Espectral indica a qual frequência a energia do espectro é centralizada. Esta feature é especialmente boa para Dubstep, onde seu centróide se localiza mais no setor dos Graves e Médios.

C. Redutores de Dimensionalidade

Ao final da extração das features, a dimensão do corpus, que foi normalizado, ficou muito alto - para cada música, 72355 valores (contados como features). Isto é devido à adequação do formato pedido pelos classificadores do sklearn. Por isso, dois métodos de redução de dimensionalidade foram inseridos ao código, para finalidade de comparação.

*1) **PCA: Principal Component Analysis - Análise de Componente Principal**:* Uma das formas mais comuns de diminuir o tempo de execução de um algoritmo de machine learning é usando o método PCA, ou Análise de Componente Principal. É um método no qual se extraem variáveis importantes (chamados de componentes) a partir de um grande set de dados em um dataset utilizando transformação linear. Com ele, temos interesse em encontrar componentes que maximizam a variância no nosso dataset. Ou seja, via PCA, projetamos um dataset completo em um subespaço diferente. Para rápido referenciamento, seguem alguns pontos-chave que é necessário saber antes de seguirmos:

- Um componente é a combinação linear das variáveis originais.
- Componentes principais são extraídas de tal forma que o primeiro componente principal explicita a máxima variância no dataset.
- O segundo componente principal tenta expor o restante da variância no dataset e não é correlacionado com o primeiro componente.
- O terceiro componente tenta expor a variância que não é exposta pelos dois primeiros componentes, etc.

Como este procedimento é feito?[9]
Resumidamente, é feito em 7 passos:

1) Selecionar o dataset completo, e desconsiderar as labels das features;
2) Computar o vetor médio d-dimensional;
3) Calcular a matriz de dispersão e a matriz de covariância;
4) Computar eigenvetores e eigenvalores correspondentes;
5) Ordenar os eigenvetores diminuindo o valor dos eigenvalores;
6) Escolher k eigenvetores com os maiores eigenvalores;
7) Transformar o dataset no novo subespaço.

É importante ressaltar que, como o PCA é um algoritmo linear, ele não vai ser capaz de interpretar relações polinomiais complexas.

*2) **t-SNE: t-distributed Stochastic Neighbor Embedding - Incorporação de Vizinhos Estocásticos t-distribuídos**:* Já o t-SNE é um algoritmo de redução de dimensionalidade não-linear que encontra padrões no dataset a partir da identificação de clusteres observados baseados na similaridade de pontos dos dados com múltiplas features.

Como este procedimento é feito?[10]
Resumidamente, é feito em 4 passos:

1) Converter as distâncias Euclideanas de alta-dimensão entre os pontos para probabilidades condicionais que representam similaridades. A similiaridade do ponto x_i ao ponto x_j é a probabilidade condicional. Para pontos próximos, a probabilidade condicional é relativamente alta.
2) Para os pontos homólogos de baixa dimensão y_i e y_j dos pontos de alta dimensão x_i e x_j, é possível computar um probabilidade condicional similar. Em termos simplificados, os passos 1 e 2 são para calcular as probabilidades condicionais em alta e baixa dimensão.
3) Medir a minimização as somas das diferenças das probabilidades condicionais.
4) Computar as variâncias σ_i da t-distribuição de Student centralizado em cada ponto de alta dimensão x_i.

É importante lembrar que, mesmo que t-SNE identifique clusteres, ele não é um algoritmo para clustering, mas sim de redução de dimensionalidade. Isso porque ele mapeia o dado multi-dimensional em um espaço de menor dimensão; isso faz com que as features de entrada não sejam mais identificáveis. Por isso, não é possível fazer inferência baseado somente no output do t-SNE.

D. Classificadores

Quanto aos classificadores, foi decidido, por influência da literatura do artigo publicado [3] e da tese relacionada à esta área [4], que usaremos o modelo

de Support Vector Machines e o modelo probabilístico Modelo de Misturas Gaussiano.

1) ***SVM - Support Vector Machines****:* Se trata de um algoritmo de classificação que se apoia na intuição inicial de que, quando as classes podem ser separadas por um hiper-plano, uma análise da ”fronteira” entre as classes pode ser efetiva para se gerar vetores de suporte que proporcionem o cálculo de tal hiperplano. A consequência disto é que, mesmo com uma grande quantidade de dados, o algoritmo poderia realizar os cálculos mais rapidamente, pois este só analisaria um sub-conjunto das observações, ou seja, só aquelas necessárias para o cálculo dos vetores de suporte.

Uma característica importante são as margens do separador. O algoritmo que inspirou o SVM, o Classificador de Margem Máxima, como o nome sugere, utilizava o raciocínio simples de que seria mais eficiente colocar seu hiperplano exatamente no meio da distância entre as duas classes. Ou seja, o mais longe da ”fronteira” de cada uma delas. O SVM estende este conceito para uma margem de tolerância, chamado parâmetro C. Basicamente o classificador estabelece um limite em que amostras de uma classe podem aparecer do outro lado. Isso dá maior robustez ao algoritmo, que passa a conseguir classificar com boa taxa de precisão, mesmo os dados que não sejam tão uniformemente separáveis. Isto possibilita aplicações eficientes para uma maior quantidade de datasets, mesmo aqueles muito grandes e não tão bem distribuídos.

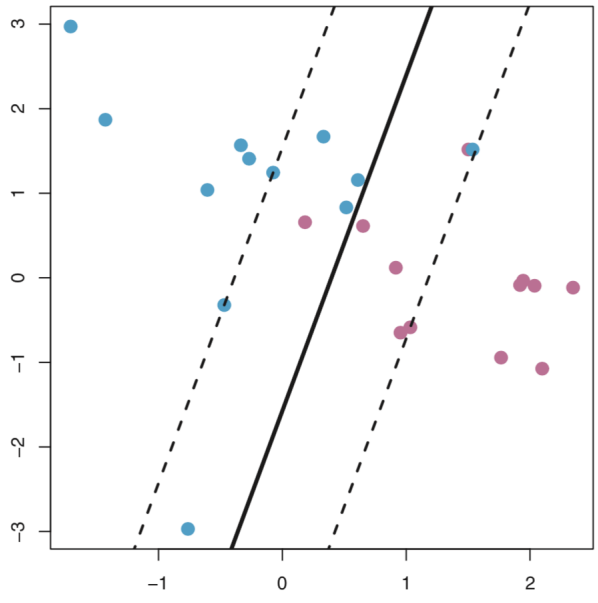


Fig. 2. Funcionamento do parâmetro C. Fonte: [2]

Mas, e quando as classes não são linearmente separáveis?

É muito comum se deparar com conjuntos de dados que não são linearmente separáveis, isto torna mais complexa a aplicação de algoritmos de aprendizado de máquina, devido ao acréscimo de complexidade inerente.

O SVM, para classificação não linear, aumenta o espaço de features, adicionando dimensões não-lineares a partir de equações de kernel. Ou seja, algumas operações não-lineares são feitas sobre as features do espaço de observações. Elas podem ser do tipo polinomial, radial, gaussiano, etc. A partir deste novo espaço de features, o algoritmo realiza a classificação linear da mesma forma que faria no espaço original. Porém, quando transposto para o espaco original, a fronteira de classificação toma feições não-lineares de acordo com as operações que foram feitas.

2) ***GMM - Modelos de Mistura Gaussiano****:* Antes, falemos um pouco sobre Maximização de Expectativa. Maximização de Expectativa é um algoritmo no qual procura-se a probabilidade máxima para parâmetros de um modelo, quando os dados estão incompletos, com pontos de dados faltando ou há variáveis latentes que estão escondidas. Utilizando estes dados incompletos, é feito uma estimação de novos sub-datasets, que são usados para criar uma nova hipótese de dados para o primeiro dataset, e o processo continua até convergir. Este algoritmo será usado no Modelo de Misturas Gaussiano.

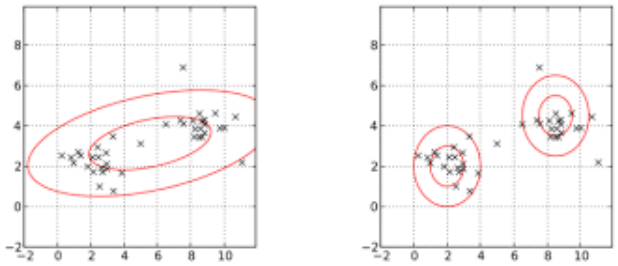


Fig. 3. Funcionamento do Modelo de Misturas Gaussiano. Á esquerda, classificação para um componente, Á direita, classificação para dois componentes. Fonte: [12]

O Modelo de Misturas Gaussiano se trata de um modelo probabilístico para representar subpopulações inseridos em uma população total. Modelos de Mistura não dependem de saber qual subpopulação um ponto de dado pertence. Isto permite que o modelo aprenda as subpopulações automaticamente. Este modelo é parametrizado por dois tipos de valores: os *pesos de*

mistura de componentes e o componente *médias* e *variâncias/covariâncias*. Para um Modelo de Mistura Gaussiano, com K componentes, o k-ésimo componente tem uma média de μ_k, uma matriz de covariância de Σ_k e um peso de mistura de componentes ϕ_k para o caso multidimensional, que é o que usamos neste trabalho. Se sabemos qual é o número de componentes K, a Maximização de Expectativa é usada para estimar os parâmetros do Modelo de Mistura. Para o Modelo de Misturas Gaussiano, este processo de maximização de expectativa consiste em dois passos:

1) Passo E - Expectativa: Calcular a expectativa das tarefas dos componentes para cada ponto de dado x_i pertencente a X, dados os parâmetros dos modelos ϕ_i, μ_k e Σ_i.
2) Passo M - Maximização: Maximizar as expectativas calculadas no passo E em respeito aos parâmetros do modelo. Consiste em atualizar os parâmetros ϕ_i, μ_k e Σ_i.

Este processo repete até o algoritmo convergir.

E. Sobre o modelo adotado para a Resolução do Problema

*1) **Pré-processamento**:*

- **Captura e Modelagem inicial do corpus**: Foram utilizadas as bibliotecas: `os`, para capturar os arquivos .mp3 por pasta e `libROSA` juntamente com `numpy`, para extrair as features de cada arquivo .mp3.
- **Normalização e Redução de Dimensionalidade**: Foram utilizadas as bibliotecas: `sklearn` para utilizar os algoritmos t-SNE e PCA no corpus, que foi normalizado com normalize()

*2) **Classificação**:* Na classificação foi utilizada a biblioteca `sklearn` para utilizar os classificadores SVM e GMM com validação cruzada 90/10 (sendo 90% de dados de treino, e 10% de teste).

III. Solução, Análise e Resultados

Seguindo o modelo adotado acima, trabalhamos da seguinte forma:

A. Pré-processamento

É importante ressaltar que a organização das pastas é como já expomos na seção ”Introdução”: Para o pré-processamento, duas funções foram criadas, para melhor organização:

- *get_features_from_mp3(folderpath, folder, file_name)*: Utilizando como parâmetro o caminho do arquivo, o nome da pasta específica e o nome do arquivo selecionado, a função faz o seguinte procedimento:
  1) O arquivo é aberto com a biblioteca `libROSA`, que permite análise de áudio em termos temporais e espectrais;
  2) As features especificadas na seção II.B - *”Features: quais features vamos utilizar?”* são extraídas uma a uma;
  3) Estas features foram agrupadas em um array 1D, que é retornado para o corpus.

  O array que é retornado vale somente para um único arquivo. Por isso, esta função é executada repetidamente até todos os arquivos .mp3 das subpastas da pasta ”data” forem visitados.
- *load_from_folder(folderpath)*: Utilizando como parâmetro o caminho absoluto para a pasta onde estão os arquivos, a função vai de pasta em pasta para selecionar cada um dos arquivos sequencialmente para extrair as features utilizando a função *get_features_from_mp3()* tendo como resultado o corpus, que consiste em:
  - *”data”*: conjunto de dados com features extraídas a partir da função *get_features_from_mp3*.
  - *”target”*: os números correspondentes às classes.
  - *”names”*: os nomes das classes. São correspondentes a cada nome de pasta inserida na pasta ”data”.

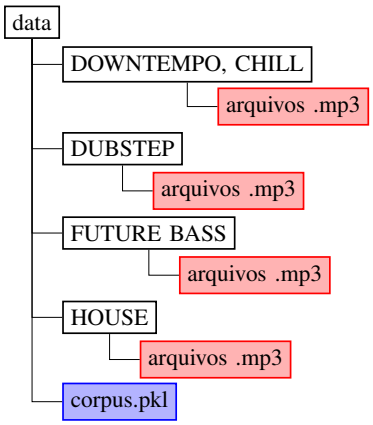


Fig. 4. Organização de diretório dos dados que serão usados para a classificação.

B. Normalização e Redução de Dimensionalidade

O corpus resultante do pré-processamento gerou uma matriz *de 323 por 77523*, ou seja, para cada arquivo extraído, 77523 valores que definem as features. Por isso,

foi necessário reduzir a dimensionalidade utilizando dois algoritmos. Os módulos da biblioteca `sklearn` que foram usadas são:

- `decomposition`, para o PCA, onde reduzimos a dimensionalidade para 10 componentes. O corpus reduzido por PCA é a variável corpus_data_pca
- `manifold`, para o t-SNE, onde reduzimos a dimensionalidade para 3 componentes usando 300 iterações para tal. O corpus reduzido por t-SNE é a variável corpus_data_tsne

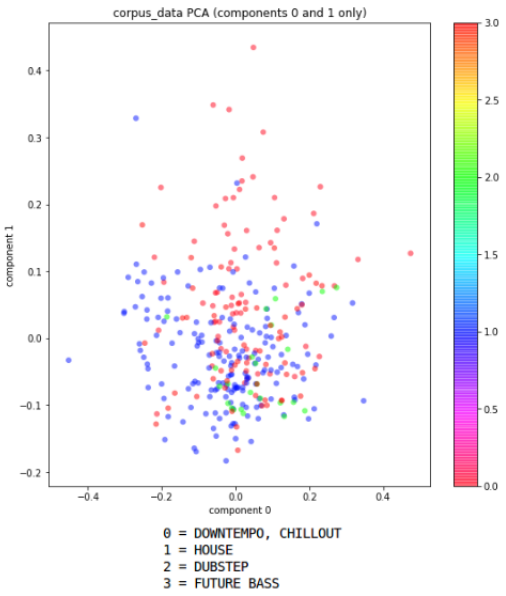


Fig. 5. Corpus reduzido utilizando PCA, mostrando dois dos dez componentes

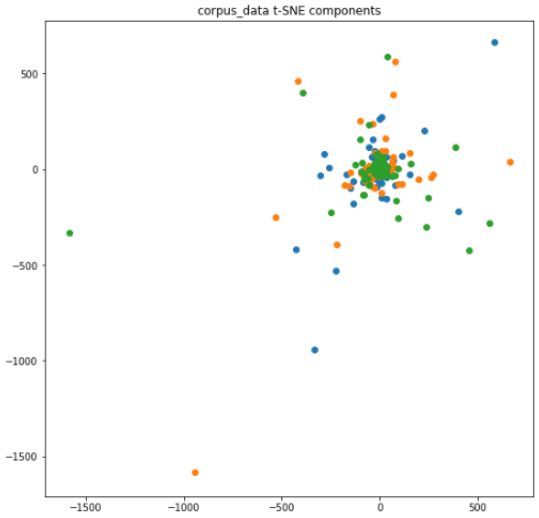


Fig. 6. Corpus reduzido utilizando t-SNE.

Podemos ver aqui que, nos dois casos, temos um caso de crowding, isto é, as classes estão relativamente agrupadas em um ou mais pontos.

C. *Classificação*

Já na classificação, utilizamos a biblioteca `sklearn` para classificar o corpus_data_pca e o corpus_data_tsne com SVM e GMM usando cross-validation 90/10 (sendo 90% dados de treino e 10% dados de teste) nestes corpus.

*1) **SVM: Para PCA (corpus_data_pca)**:*

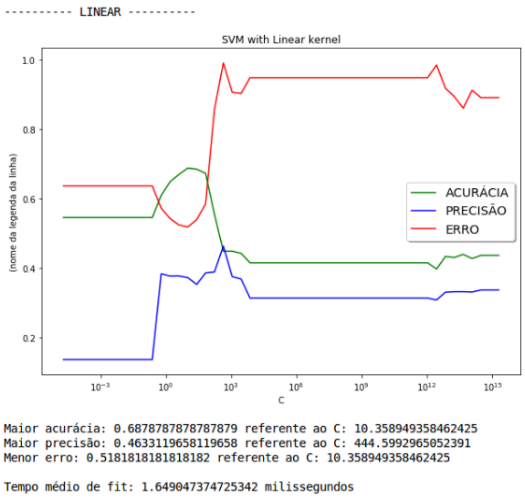


Fig. 7. Classificação SVM Linear com corpus PCA

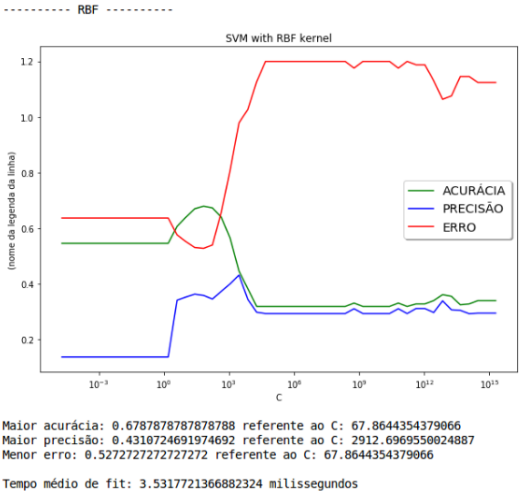
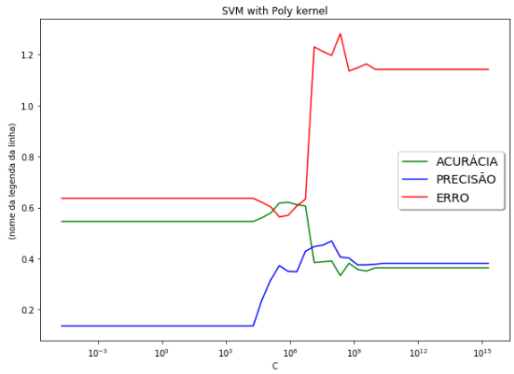


Fig. 8. Classificação SVM de Função de Base Radial com corpus PCA

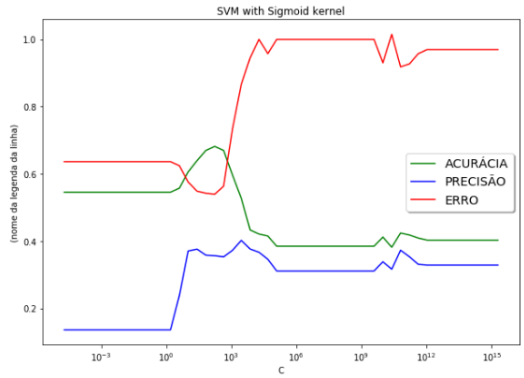
```
---------- POLY ----------
```




```
Maior acurácia: 0.6212121212121212 referente ao C: 818983.0124760838
Maior precisão: 0.4691632613817212 referente ao C: 89968653.37938906
Menor erro: 0.5636363636363636 referente ao C: 319971.74392121146

Tempo médio de fit: 2.062314033508301 milissegundos
```

Fig. 9. Classificação SVM Polinomial com corpus PCA

```
---------- SIGMOID ----------
```



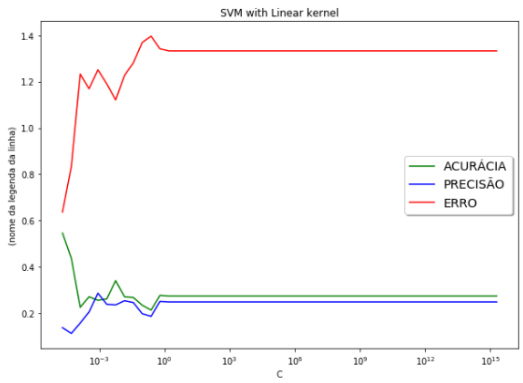

```
Maior acurácia: 0.6818181818181818 referente ao C: 173.70227475027042
Maior precisão: 0.402249307709834 referente ao C: 2912.6969550024887
Menor erro: 0.5393939393939394 referente ao C: 173.70227475027042

Tempo médio de fit: 2.9366917610168457 milissegundos
```

Fig. 10. Classificação SVM Sigmóide com corpus PCA

Classificando o corpus_data_pca com SVM, vimos um resultado pouco satisfatório, com uma média (em relação aos kernels usados) de acurácias de 66,7% e precisão de 44,1% . Mas ainda assim, o erro médio mostrado em relação aos kernels usados é de 53,7%. Isto acontece devido ao problema de crowding que vimos na normalização.

Para t-SNE (corpus_data_tsne):

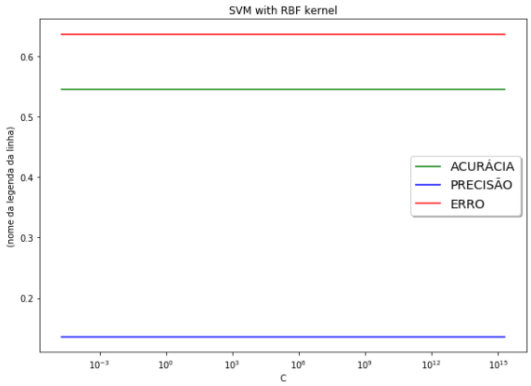
```
---------- LINEAR ----------
```




```
Maior acurácia: 0.5454545454545454 referente ao C: 2e-05
Maior precisão: 0.2851349165540341 referente ao C: 0.0008583868520257557
Menor erro: 0.6363636363636364 referente ao C: 2e-05

Tempo médio de fit: 1.2052664756774902 milissegundos
```

Fig. 11. Classificação SVM Linear com corpus t-SNE

```
---------- RBF ----------
```




```
Maior acurácia: 0.5454545454545454 referente ao C: 2e-05
Maior precisão: 0.13636363636363635 referente ao C: 2e-05
Menor erro: 0.6363636363636364 referente ao C: 2e-05

Tempo médio de fit: 4.633181095123291 milissegundos
```

Fig. 12. Classificação SVM de Função de Base Radial com corpus t-SNE

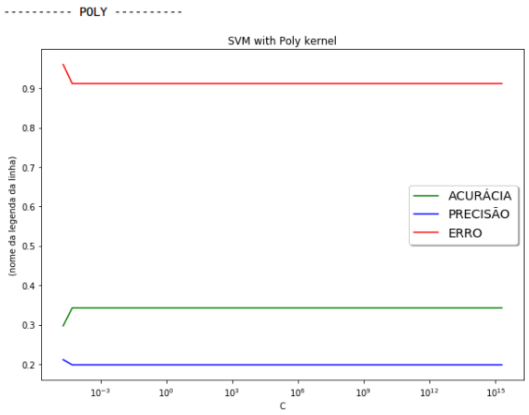


Fig. 13. Classificação SVM Polinomial com corpus t-SNE

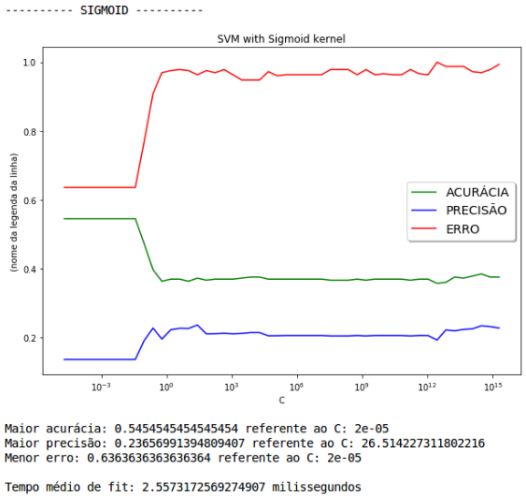


Fig. 14. Classificação SVM Sigmóide com corpus t-SNE

Classificando o corpus_data_tsne com SVM, vimos um resultado muito insatisfatório, com uma média (em relação aos kernels usados) de acurácias de 49,4% e precisão de 21,7% . E o erro médio mostrado em relação aos kernels usados é de 70,5%. Isto acontece devido ao problema de crowding que vimos na normalização, além de que o redutor de dimensionalidade t-SNE é não-linear. Isto fez com que o crowding fosse mais acentuado no corpus_data_tsne.

2) GMM: Para PCA (corpus_data_pca):

```
-------- PCA --------
Maior acurácia: 0.26666666666666666
Maior precisão: 0.06666666666666667
Menor erro: 1.0212121212121212

Tempo médio de fit: 4.949140548706055 milissegundos
```

Fig. 15. Resultado da classificação GMM para corpus PCA

Classificando o corpus_data_pca com GMM, vimos um resultado muito insatisfatório, como mostrado na figura acima. Isto acontece devido ao problema de crowding que vimos na normalização. Com o algoritmo PCA, tivemos um caso de crowding também, isto é, os pontos dos dados se encontram agrupados em um mesmo lugar. Por isso e por causa do comportamento do classificador GMM, o erro maximizou.

Para t-SNE (corpus_data_tsne):

```
-------- t-SNE --------
Maior acurácia: 0.3090909090909091
Maior precisão: 0.12942798395523647
Menor erro: 1.0454545454545454

Tempo médio de fit: 4.220271110534668 milissegundos
```

Fig. 16. Resultado da classificação GMM para corpus t-SNE

Classificando o corpus_data_tsne com GMM, vimos um resultado muito insatisfatório, como mostrado na figura acima. Isto acontece devido ao problema de crowding que vimos na normalização, além de que o redutor de dimensionalidade t-SNE é não-linear. Isto fez com que o crowding fosse mais acentuado no corpus_data_tsne. E por causa do comportamento do classificador GMM, o erro de fato maximizou.

IV. Trabalhos Futuros

Os trabalhos futuros serão principalmente focados no pré-processamento e captura de features. Como na música eletrônica as divisões são bem definidas em sua estrutura musical (geralmente a estrutura musical de uma música eletrônica segue, sequencialmente: introdução, verso 1, build-up, drop 1, beatdown, verso 2, build-up, drop 2, outro), podemos capturar faixas nas músicas tendo em mente estas estruturas musicais utilizando de beat-tracking para tal. Por exemplo: capturar um drop de Future Bass e um drop de Dubstep. As estruturas musicais que seriam capturadas seriam o build-up e o drop principalmente, porque neles existem a maior parte de energia em uma música.

V. Conclusões

O assunto sobre classificação de subgêneros de Música Eletrônica que abordamos é muito interessante, mas também pode trazer muitos desafios. Quando se trata de manipulação de áudio, o pré-processamento deve ser extremamente sofisticado, uma vez que existem músicas de várias naturezas dentro e fora do gênero "Música Eletrônica". Com os resultados da redução de dimensionalidade e das classificações, podemos ver a importância disto, já que, mesmo que as features tenham sido extraídas de uma forma relativamente satisfatória, era necessário ser mais minucioso. Por isso, como dito na seção "Trabalhos Futuros", fazer uma captura por beat-tracking relacionando as estruturas musicais com as features pode ser satisfatório.

References

[1] Repositório do Projeto: https://github.com/arthurveiga/ml_edm_subgenre_classification

[2] Biblioteca Python: LibROSA - https://librosa.github.io/librosa/index.html

[3] Artigo: Zhiwei Gu, Li Guo, Tianchi Liu - Music Genre Classification via Machine Learning [2017] (http://cs229.stanford.edu/proj2017/final-reports/5244969.pdf)

[4] Artigo: Chang-Hsing Lee, Jau-Ling Shih, Kun-Ming Yu, and Jung-Mau Su - Automatic Music Genre Classification using Modulation Spectral Contrast Feature

[5] Tese: Austin C. Chen - Automatic Classification of Electronic Music and Speech/Music Audio Content [2014]

[6] Artigo: Daniel Gärtner, Christoph Zipperle, Christian Dittmar - Classification of Electronic Club Music [2010]

[7] Artigo: George Tzanetakis, Georg Essl, Perry Cook - Automatic Musical Genre Classification of Audio Signals [2002]

[8] Spectral Features - Music Information Retrieval: https://musicinformationretrieval.com/spectral_features.html

[9] Sebastian Raschka - PCA In Depth: https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#generating-some-3-dimensional-sample-data

[10] Analytics Vidhya - t-SNE: https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/

[11] An Introduction to Statistical Learning - Gareth James, Daniela Witten; Springer NYH, Dordrecht London

[12] Modelo de Misturas Gaussiano - GMM: https://brilliant.org/wiki/gaussian-mixture-model/

[13] Music Information Retrieval - https://musicinformationretrieval.com/