UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA

Continuação de playlists usando filtragem colaborativa

EEL 410250 - Aprendizado de Máquina
Gustavo de Paula Santos

1. INTRODUÇÃO

Este projeto tem como objetivo desenvolver um sistema que realize a continuação de playlists usando filtragem colaborativa com base em dados de playlists do Spotify.

O problema de continuação de playlists, chamado de *automatic playlist continuation* (APC), é um problema de sistema de recomendação, em que é necessário ordenar, dentro de um conjunto de amostras possíveis, quais são as amostras mais prováveis de serem as corretas para um determinado cenário, i.e. quais as músicas mais prováveis de serem selecionadas por um usuário para compor uma playlist criada por ele.

O objetivo do modelo é, então, selecionar, dentre as músicas possíveis, quais músicas combinam com uma playlist inicializada por um usuário.

1.1. Trabalhos relacionados

Por se tratar de um problema introduzido no desafio da conferência RecSys de 2018 (Chen *et al.* 2018). Há uma série de trabalhos que buscaram resolver este mesmo problema, uma lista deles, bem como uma análise das melhores soluções encontradas, está disponível no artigo escrito por Zamani e outros (2019).

Dentre as soluções encontradas, este trabalho selecionou a encontrada pela equipe *hello world!* da trilha principal do desafio para se inspirar. Para a resolução do desafio, a equipe utilizou uma arquitetura multimodal de filtragem colaborativa, que chamaram de MMCF (do inglês *Multimodal Collaborative Filtering*) (Yang *et al.*, 2018).

A arquitetura usada pela equipe consiste num *ensemble* de duas arquiteturas diferentes. Para a informação da presença ou não de uma música ou artista na playlist, usaram uma arquitetura de *Denoising Autoencoders* usando estratégia *Hide and Seek*, que oculta parte das informações durante o treinamento para que a

reconstrução do autoencoder seja feita mesmo que algumas informações estejam ocultas na playlist.

Para usar a informação do nome da playlist, aplicaram uma arquitetura CharCNN. Como a solução usada neste trabalho não usou a informação do título da playlist, não se detalhou a descrição desta arquitetura.

2. METODOLOGIA

Explicam-se abaixo os materiais e métodos usados no desenvolvimento do trabalho.

2.1. Dados

Os dados usados para o desenvolvimento deste projeto são advindos do dataset criado especificamente para o desafio de APC. O nome dado ao *dataset* foi *Million Playlist Dataset* (MPD) (Chen *et al.,* 2018).

O *dataset* consiste em vários arquivos JSON, chamados de *slices*, contendo um array de 1.000 (mil) *playlists* e as músicas dentro de cada *playlist*. Cada *playlist* contém dados e metadados associados a ela. As descrições das informações disponíveis para cada playlist estão descritas abaixo:

- **pid:** ID da playlist;
- **name:** nome da playlist;
- **description:** descrição da playlist, fornecida pelo usuário. Na data de coleta dos dados era uma feature nova no Spotify, então muitas das playlists não têm essa informação;
- **modified_at:** timestamp da meia noite (GMT) do dia em que a playlist foi editada a última vez;
- **num_artists:** número total de artistas únicos presentes na playlist;
- **num_albums:** número total de álbuns únicos presentes na playlist;

- **num_tracks:** número de músicas presentes na playlist;
- **num_followers:** número de seguidores da playlist no momento de criação do MPD;
- **num_edits:** quantidade de sessões de edição. Músicas adicionadas dentro de um intervalo de 2h são consideradas adicionadas em uma única sessão de edição;
- **duration_ms:** duração total da playlist, em milissegundos
- **collaborative:** se a playlist é ou não colaborativa, i.e. múltiplos usuários podem editar a playlist;
- **tracks:** um array de músicas que estão na playlist. Cada elemento do array tem os seguintes dados:
    - **track_name:** nome da música;
    - **track_uri:** o URI da música no Spotify;
    - **album_name:** nome do álbum em que a música está;
    - **album_uri:** o URI do álbum no Spotify;
    - **artist_name:** nome do artista principal da música;
    - **artist_uri:** URI do artista principal no Spotify;
    - **duration_ms:** duração da música em millissegundos
    - **pos:** posição da música na playlist (iniciando em 0)

  Um exemplo de entrada de um arquivo JSON está ilustrado abaixo.

```
{
        "name": "musical",
        "collaborative": "false",
        "pid": 5,
        "modified_at": 1493424000,
        "num_albums": 7,
        "num_tracks": 12,
        "num_followers": 1,
        "num_edits": 2,
        "duration_ms": 2657366,
        "num_artists": 6,
        "tracks": [
            {
                "pos": 0,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
                "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Finalement",
                "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
                "duration_ms": 166264,
                "album_name": "Dancing Chords and Fireflies"
            },
            {
                "pos": 1,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:23EOmJivOZ88WJPUbIPjh6",
                "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Betty",
                "album_uri": "spotify:album:3lUSlvjUoHNA8IkNTqURqd",
                "duration_ms": 235534,
                "album_name": "Endless Smile"
            },
```


Devido à limitação de poder computacional e tempo, selecionou-se apenas as 10.000 primeiras playlists para o desenvolvimento do projeto. Portanto, todas as

informações que se seguem são considerando apenas o conjunto das playlists selecionadas.

Além disso, usou-se somente a informação da presença ou não de uma faixa na playlists, como se descreve a seguir, portanto, utilizou-se somente a informação "pid"e "track_uri" das playlists selecionadas.

No conjunto selecionado haviam 10.000 playlists únicas e 170.089 músicas únicas. Sendo que haviam 664.712 pares (pid, track_uri).

2.2. Pré-processamento de dados

Com os dados extraídos das playlists em um pandas.DataFrame(), em que cada linha representa uma música associada a uma playlist, foi possível realizar alguns pré-processamentos.

De início, apesar de ser uma ocorrência possível na plataforma Spotify, excluíram-se aparições duplicadas, isto é, músicas que apareciam mais de uma vez na mesma playlist foram removidas de forma a manter somente uma aparição.

Continuou-se o processamento avaliando a quantidade de aparições mínimas de uma música no dataset e a quantidade mínima de músicas para uma playlist ser considerada no desenvolvimento do projeto.

Excluíram-se as músicas que apareciam em menos de 5 playlists diferentes e playlists que continham menos de 7 músicas diferentes. Ao final da etapa de limpeza, restaram 21.771 músicas e 8.885 playlists, totalizando 442.411 pares (pid, track_uri)

Para utilização em modelos de aprendizado profundo, foi necessário realizar a codificação númerica das músicas restantes no conjunto e a recodificação das playlists restantes, de forma a ter IDs das playlists que formassem uma sequência

contínua. Para isto, utilizou-se o codificador sklearn.preprocessing.OrdinalEncoder() com seus parâmetros padrão.

2.3. Split treino-validação-teste

Apesar de cada arquitetura ter suas nuances relacionadas à forma como cada split foi utilizado durante as etapas de treino e teste, os pontos em comum serão descritos nesta seção.

Como forma de simular o funcionamento de um sistema de continuação de playlists, foi necessário selecionar playlists que teriam parte de suas músicas ocultadas durante o processo de inferência, de forma que as músicas ocultas de cada playlists serviriam de *ground truth* para o cálculo de desempenho do modelo.

Selecionou-se, aleatoriamente, 72% das playlists para comporem o conjunto de treino, 20% das playlists para comporem o conjunto de teste e 8% das playlists para compor o conjunto de validação, usado para seleção de hiperparâmetros.

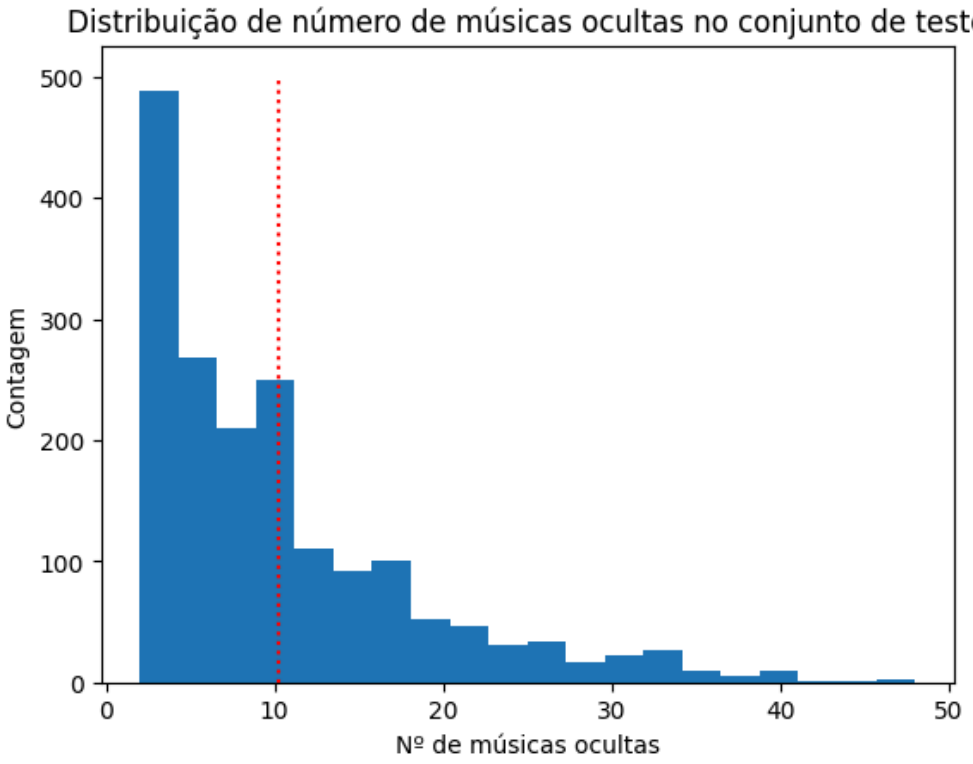
A divisão final resultou nas seguintes quantidades de músicas únicas em cada conjunto: 21.766 músicas presentes no conjunto de treino, 12.091 músicas presentes no conjunto de validação e 18.735 músicas presentes no conjunto de teste. Verificou-se se a divisão realizada resultava num conjunto treino+validação, a ser usado no treinamento final, que tivesse todas as músicas possíveis do conjunto presentes.

Nos conjuntos de treino e validação, para cada playlist do conjunto, selecionou-se, também aleatoriamente, 20% das músicas para serem ocultadas durante a etapa de inferência e avaliação.

Para garantir que os resultados obtidos em modelos diferentes fossem comparáveis, usou-se essa mesma divisão nos diferentes treinamentos, adaptando-a para cada arquitetura.

A figura 1, abaixo, ilustra a distribuição da quantidade de músicas ocultadas no conjunto de teste. A média, representada pela linha vermelha, tem valor igual a 10,226

Figura 1 - Distribuição de músicas ocultas nas playlists do teste



2.4. Arquiteturas utilizadas

Comparou-se o desempenho de duas diferentes arquiteturas: Fatoração Matricial e Autoencoder de redução de ruído (DAE, do inglês *denoising autoencoder*). Cada uma com suas peculiaridades em relação à etapa de treinamento.

2.4.1. FATORAÇÃO MATRICIAL

Para o modelo de fatoração matricial, faz-se necessário que cada playlist seja apresentada ao modelo com exemplos de músicas positivas e negativas (aqui

representando presentes e não presentes na playlist). As entradas do modelo durante o treinamento são trios do tipo (pid, track_id, presente).

Com isto, as playlists selecionadas para o conjunto de treino da seção 2.3 tiveram todas as suas músicas entregues ao modelo como trios (pid, track_id, 1). Para a composição da parte negativa dessas playlists, selecionou-se, aleatoriamente e para cada playlist, a mesma quantidade de músicas presentes no rol de músicas possíveis. Isto é, se uma playlists do treino tem 30 músicas, 30 outras músicas foram selecionadas para aparecer no treinamento como negativas, na forma de um trio (pid, track_id, 0).

Tanto para o conjunto de validação quanto para o conjunto de teste, foram entregues ao modelo, como positivas, durante o treinamento, apenas as músicas que não foram selecionadas para serem escondidas. Da mesma forma que ocorreu nas playlists do conjunto de treino, foram selecionadas músicas para aparecem como negativas em cada playlist com músicas ocultas.

O treinamento foi realizado com estes trios usando a função perda de entropia cruzada binária.

Para a inferência, tanto no conjunto de validação quanto no de teste, pediu-se ao modelo que fornecesse a predição para todas as combinações (pid, track_id) que não apareceram como positivas durante o treinamento. A avaliação foi realizada observando as predições para as músicas ocultas, usando as métricas de avaliação que se explica a seguir.

Realizou-se a avaliação de hiperparâmetros testando diferentes valores de *batch size* e taxa de aprendizado. Os valores testados para *batch sizes* foram 128, 256 e 512 e para taxa de aprendizado foram 0,001, 0,01 e 0,1. Todos os treinamentos foram feitos usando 20 épocas.

Os melhores hiperparâmetros encontrados no conjunto de validação foram *batch size* igual a 256 e taxa de aprendizado igual a 0,01.

2.4.2. AUTOENCODER DE REDUÇÃO DE RUÍDO

Cada playlist de entrada no modelo DAE é representada por um vetor com entradas binárias de dimensão igual à quantidade total de músicas possíveis, neste caso, 21.771. Cada entrada do vetor representa a presença (1) ou não (0) da música de ID igual ao índice da entrada.

Neste caso, são fornecidas ao modelo durante o treinamento de validação apenas os vetores que representam as playlists separadas na seção 2.3 como conjunto de treino.

As playlists de validação e teste têm apenas as músicas não escondidas como 1 no vetor de representação, a versão completa de cada vetor foi usada somente na avaliação.

Para o treinamento final, as playlists completas do conjunto de validação foram adicionadas ao conjunto de treinamento.

A rede DAE foi composta por apenas uma camada intermediária de dimensão 64 e saída ReLU. Além disso, utilizou-se uma camada Dropout após a camada de entrada, com taxa 20%, para simular a entrada de playlists com entradas faltantes, isto é, a playlist de entrada é corrompida e o objetivo da rede é reconstruir a playlist não corrompida.

O treinamento foi feito usando a função de entropia cruzada binária como função custo.

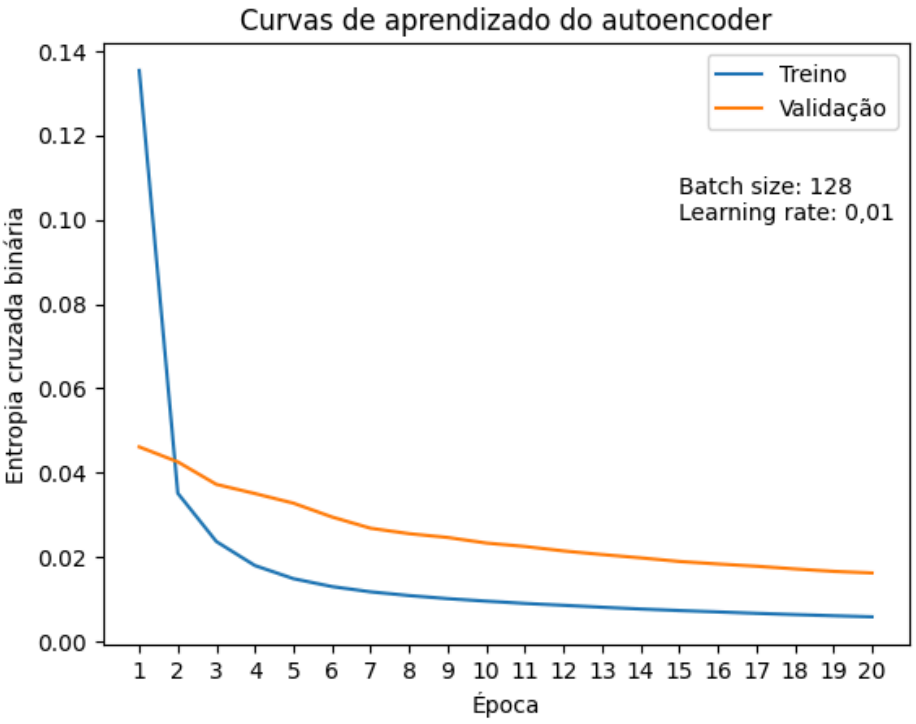
Assim como na fatoração matricial, realizou-se a avaliação de hiperparâmetros testando diferentes valores de *batch size* e taxa de aprendizado. Os

valores testados para *batch sizes* foram 128, 256 e 512 e para taxa de aprendizado foram 0,001, 0,01 e 0,1. Todos os treinamentos foram feitos usando 20 épocas.

Os melhores hiperparâmetros encontrados no conjunto de validação foram *batch size* igual a 128 e taxa de aprendizado igual a 0,01.

Abaixo, apresentam-se as curvas de aprendizado para a etapa de otimização de hiperparâmetros do autoencoder. As curvas foram produzidas usando os melhores hiperparâmetros definidos e representam a função perda entropia cruzada binária.

Figura 2 - Curvas de aprendizado do autoencoder



Pela análise das curvas, é possível perceber um início de overfitting. No entanto, percebe-se uma tendência negativa da curva de validação, ou seja, haveria espaço para melhora com o aumento do número de épocas.

2.5. Métricas de avaliação

As métricas de avaliação foram definidas e selecionadas de acordo com a ideia do projeto. Foram adaptadas duas das três métricas de avaliação do desafio original. São elas: recall e clicks. Uma explicação mais detalhada de cada métrica está a seguir.

2.5.1. RECALL

O recall, comumente traduzido como sensibilidade, é uma medida numérica que avalia a quantidade de elementos relevantes que foram selecionados. Isto é, de todas as amostras que deveriam ser selecionadas para continuar uma playlist, quantas de fato foram selecionadas.

O recall foi calculado de forma que as N músicas mais relevantes, de acordo com o modelo, compunham o conjunto de amostras preditas positivas. N, neste caso, é a quantidade de músicas ocultas na playlist. Desta forma, calculou-se o recall da seguinte maneira.

$$Recall = \frac{|\hat{y} \cap y|}{|y|}$$

Em que $\hat{y}$ é o conjunto de músicas preditas como as mais relevantes para continuar a playlist de mesmo tamanho que |y|.

Como, para cada falso positivo, i.e. uma música que não foi ocultada na playlist sendo selecionada para compor a playlist, há um falso negativo, i.e. uma música oculta na playlist que não foi selecionada para continuar a playlist, o valor numérico do recall é igual ao valor da precisão.

2.5.2. CLICKS

A métrica recebe este nome por conta da funcionalidade do Spotify de permitir que o usuário carregue músicas relevantes em grupos de 10 sugestões. A ideia da métrica seria simular a quantidade de clicks que um usuário teria que dar até encontrar uma música relevante para a playlist.

Traduzindo para o problema tratado neste trabalho, e para simplificar o cálculo, adaptou-se a métrica para simplesmente a posição da primeira música relevante para a playlist, ou seja, a posição da música oculta mais alta no ordenamento de relevância do modelo.

3. RESULTADOS

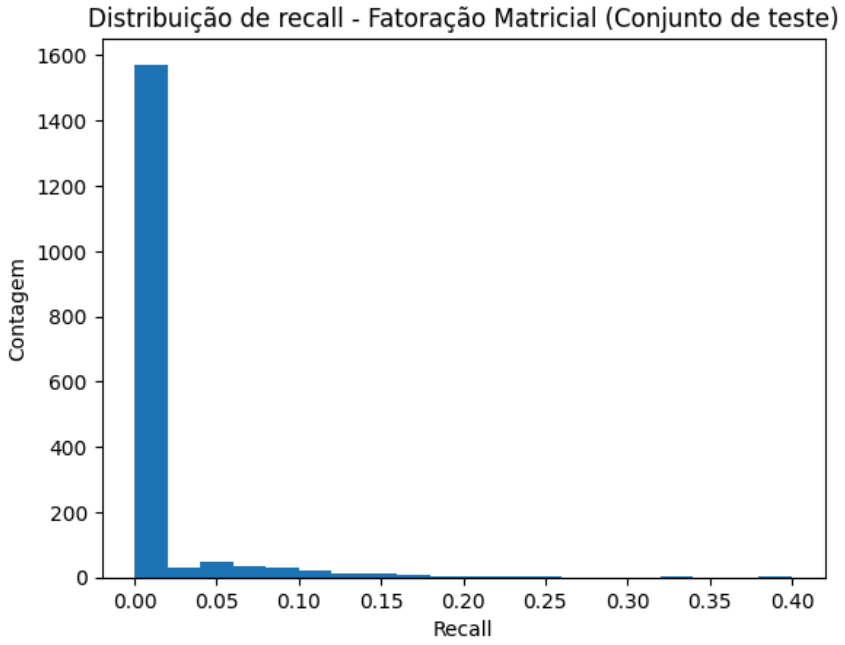
Considerando o objetivo deste trabalho como comparação entre as duas arquiteturas apresentadas anteriormente. Apresentam-se abaixo os resultados para cada uma das arquiteturas avaliadas.

3.1. Fatoração matricial

O recall médio encontrado no conjunto de teste foi de 0,010, já a média da métrica clicks foi 1.326,22.

Para a análise de distribuição, ilustra-se abaixo, na figura 3, o histograma dos recalls no conjunto de teste.

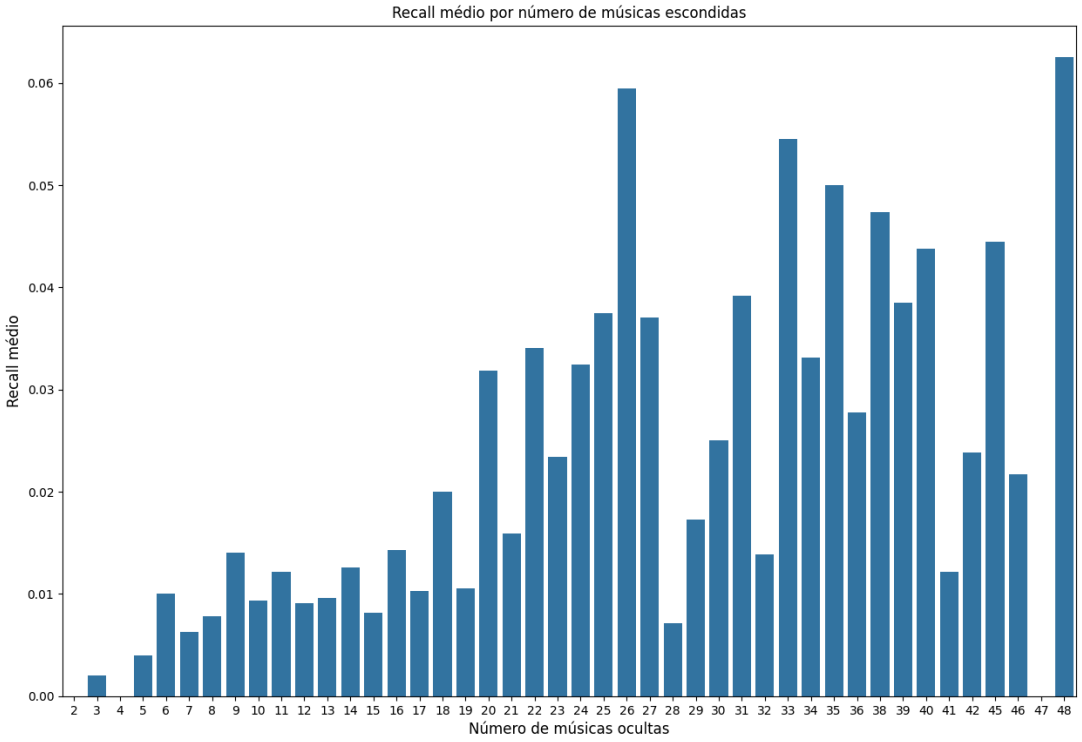
Figura 3 - Distribuição de recall na fatoração matricial



É possível perceber que grande parte dos recalls usando fatoração matricial está no bin próximo a 0.0, ou seja, para a maioria das playlists no conjunto de teste, o modelo não aponta nenhuma música dentre as ocultas como as mais relevantes.

A fim de verificar se o resultado não é influenciado pela quantidade de músicas ocultadas, ou seja, verificar se a tarefa não está difícil demais para o modelo, realizou-se uma análise em função da quantidade de músicas escondidas. Esta análise está ilustrada na figura 4.

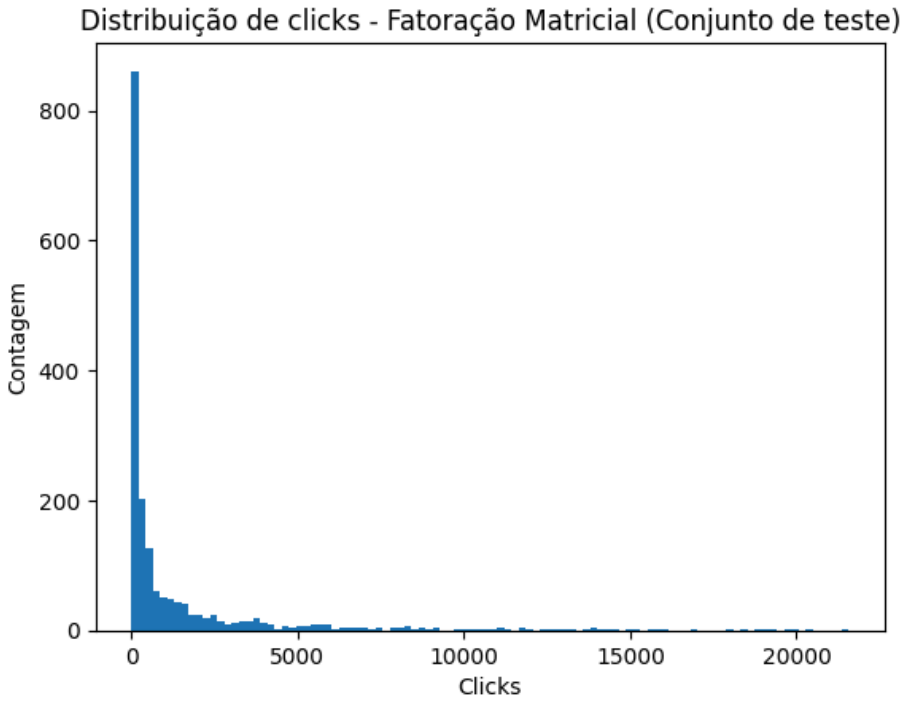
Figura 4 - Recall médio por número de músicas escondidas



Apesar de haver um aumento na performance em função do aumento do número de músicas escondidas, percebe-se que o valor médio mais alto ainda é bastante baixo, chegando a no máximo 0,06, aproximadamente.

A métrica *clicks* pode auxiliar a entender a quantidade de músicas selecionadas como relevantes necessárias para uma música que de fato compõe a playlist aparecer. A distribuição está ilustrada abaixo, na figura 5.

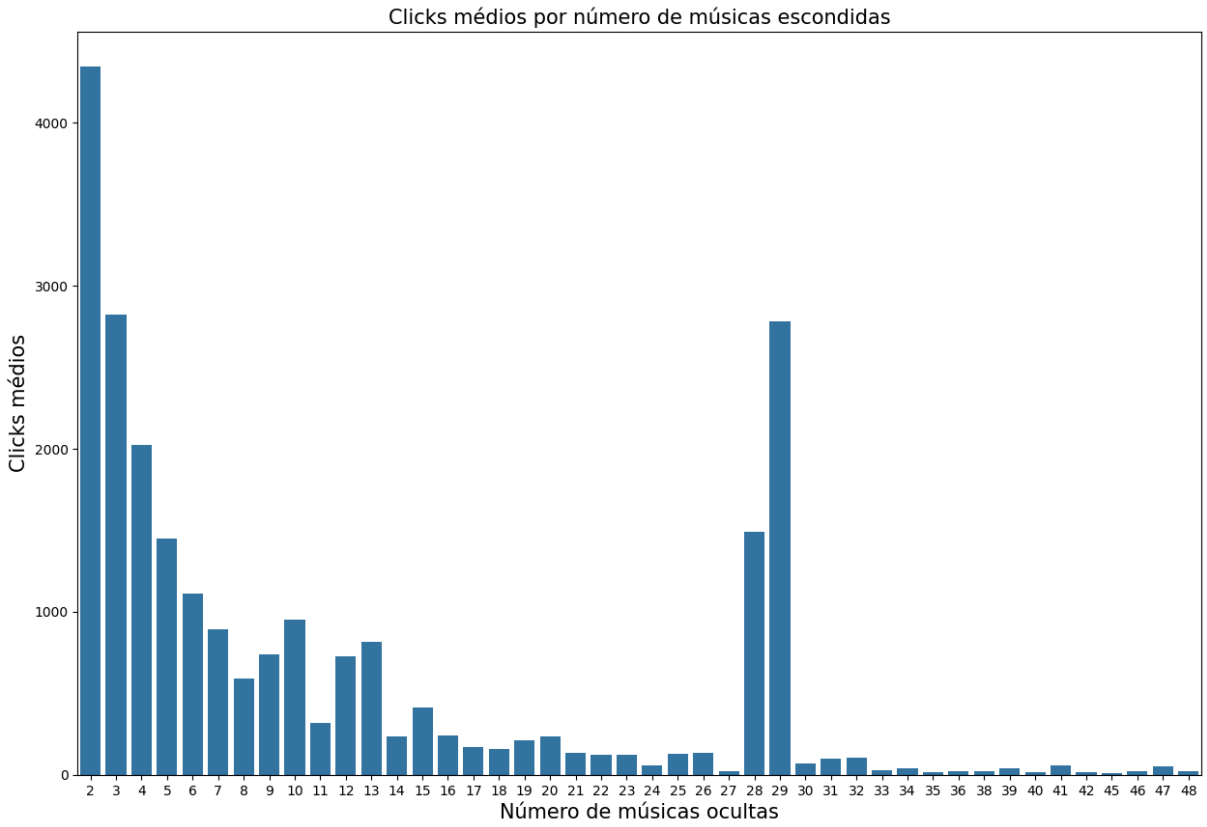
Figura 5 - Distribuição de clicks na fatoração matricial



É possível observar que há uma grande quantidade de playlists no teste que a quantidade de clicks necessários se aproximou de 0, a média, portanto, está sendo bastante influenciada pelos outliers, ilustrados no histograma acima.

Assim como para o recall, é interessante analisar se a quantidade de músicas ocultas surte um efeito na métrica. Esta análise está ilustrada na figura 6.

Figura 6 - Clicks médios por número de músicas escondidas



Para a análise do gráfico acima, é interessante lembrar que a métrica clicks deve ser minimizada, ao contrário da métrica recall. Outro ponto importante é a distribuição da quantidade de músicas ocultas, ilustrada na figura 1.

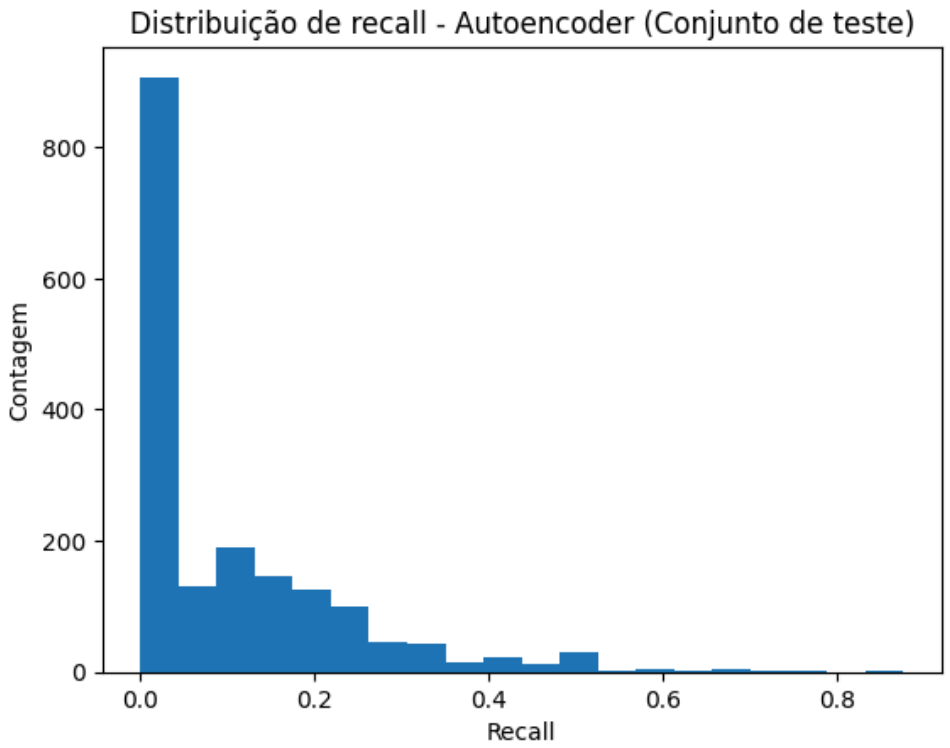
Assim como ocorre no recall, o desempenho do modelo é melhor à medida que o número de músicas ocultas aumenta. Isto pode ser justificado pelo fato de que o rol de músicas consideradas como corretas aumenta à medida que o número de músicas ocultas aumenta.

3.2. Autoencoder

Usando a arquitetura de autoencoder, o recall médio encontrado no conjunto de teste foi de 0,096 e a quantidade média de clicks foi de 145,95.

Para a análise de distribuição, ilustra-se abaixo, na figura 7, o histograma dos recalls no conjunto de teste.

Figura 7 - Distribuição de recall no autoencoder

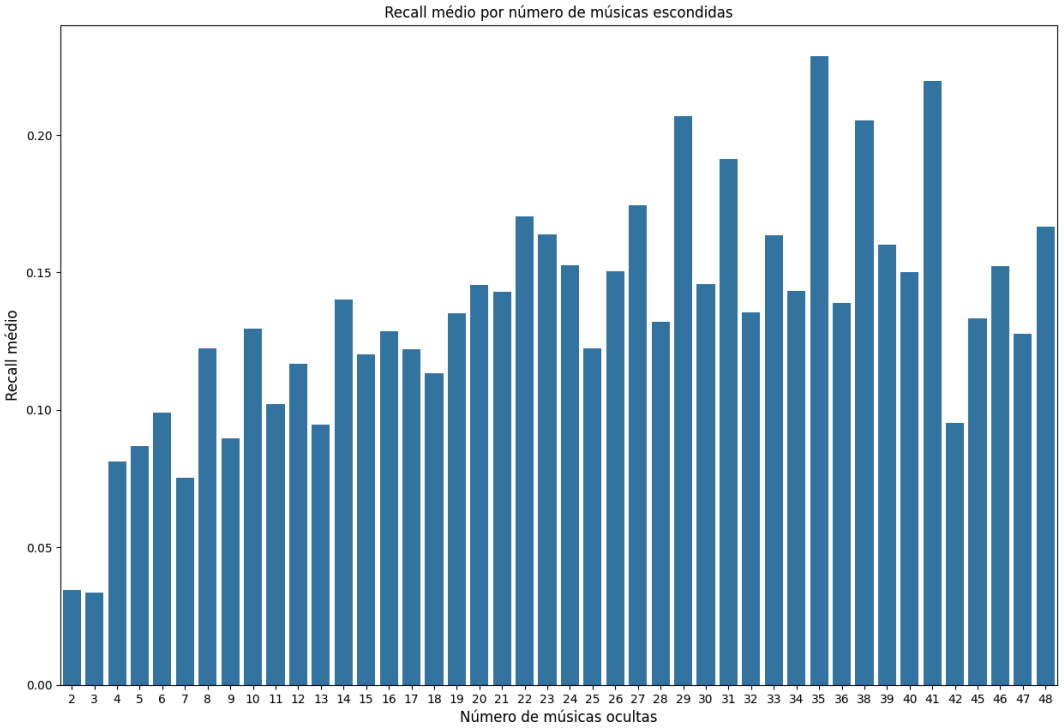


É possível perceber que, assim como na fatoração matricial, grande parte dos recalls usando fatoração matricial está no bin próximo a 0.0, ou seja, para a maioria das playlists no conjunto de teste, o modelo não aponta nenhuma música dentre as ocultas como as mais relevantes.

Há, no entanto, uma diferença na quantidade de amostras para valores mais altos, por exemplo, o recall mais alto na fatoração matricial foi de 0,4 enquanto que no autoencoder o maior recall foi de 0,877.

A fim de verificar a influência da quantidade de músicas ocultas no recall do autoencoder, realizou-se a análise ilustrada na figura 8.

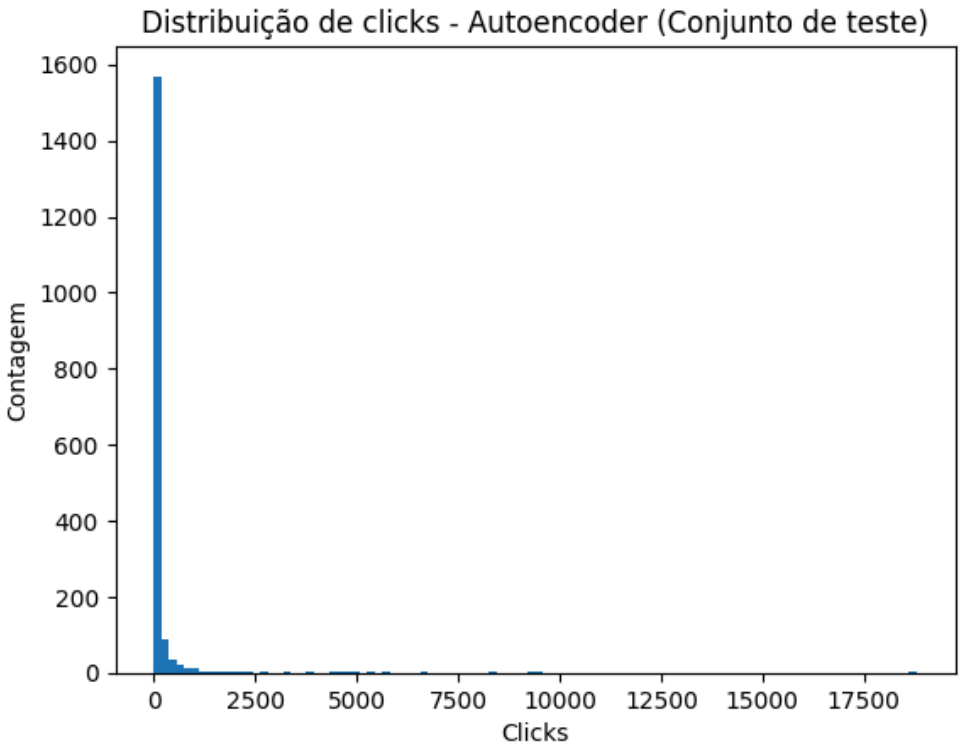
Figura 8 - Recall médio por número de músicas escondidas



Diferentemente do resultado encontrado usando a fatoração matricial. No caso do autoencoder, há um aumento significativo no desempenho à medida que se aumenta o número de músicas ocultas em cada playlist.

A métrica clicks, novamente, será uma métrica indicativa da qualidade das predições realizadas. A figura 9, abaixo, mostra a distribuição da métrica para o conjunto de teste usando o autoencoder.

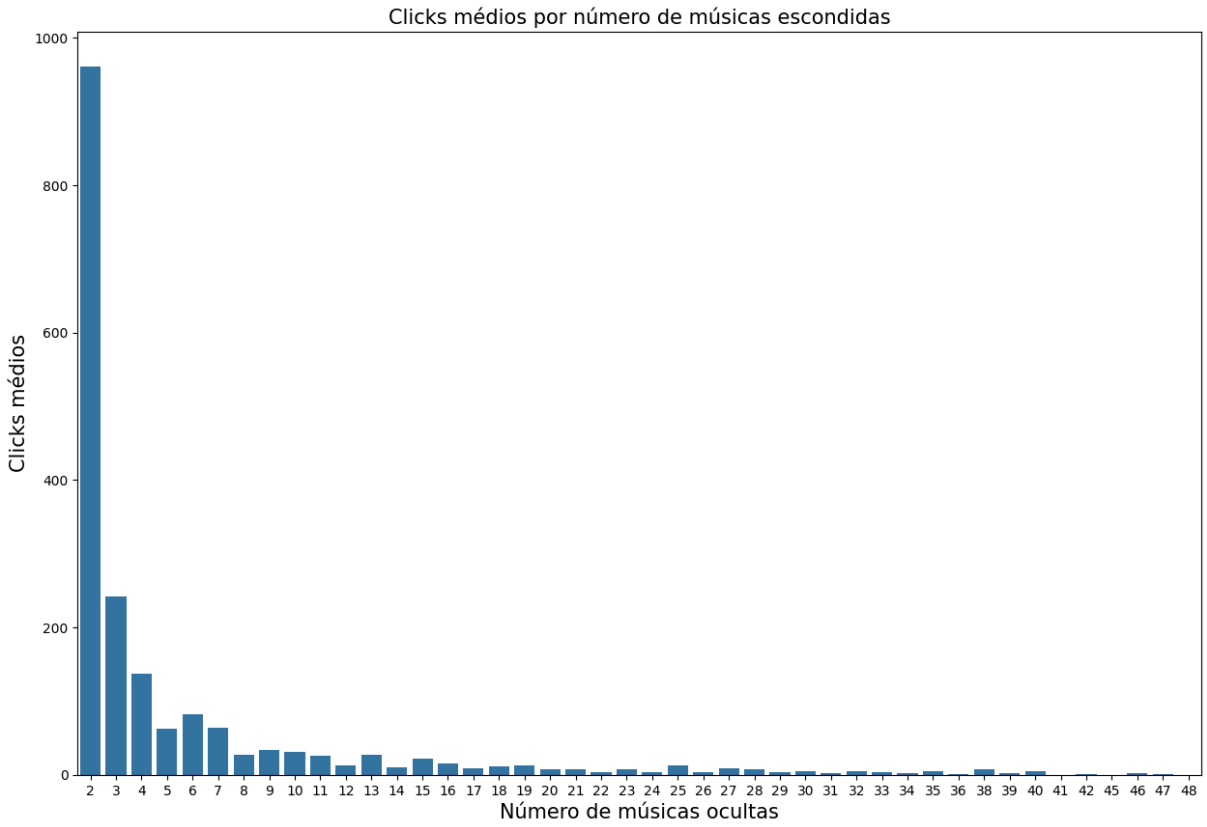
Figura 9 - Distribuição de clicks usando a arquitetura autoencoder



Há dois pontos que devem chamar a atenção na comparação entre a figura 9 e a figura 5. Primeiramente, a quantidade máxima de clicks na arquitetura autoencoder foi próxima de 17.500, enquanto na fatoração matricial foi próxima de 20.000. Segundo, a quantidade de playlists no primeiro bin, mais próximo de zero, na arquitetura autoencoder é pouco menos do que o dobro da quantidade no mesmo bin na fatoração matricial.

Bem como foi realizado para o recall e na análise de desempenho da fatoração matricial, apresentam-se abaixo, o número médio de clicks em função da quantidade de músicas ocultas na playlist. Esta análise está ilustrada na figura 10

Figura 10 - Clicks médios por número de músicas escondidas no autoencoder



Assim como observado no recall, o desempenho aumenta à medida que se aumenta a quantidade de músicas ocultas, possivelmente por aumentar a quantidade de músicas consideradas como relevantes para a playlist.

Também seguindo o observado no recall, o desempenho se mostrou bastante superior usando a estrutura autoencoder, em comparação com a fatoração matricial.

O fato de a métrica clicks representar melhor a mudança de comportamento à medida que se aumenta o número de músicas ocultas chama a atenção para um fato. Uma música não fazer parte do conjunto oculto não, necessariamente, significa que ela não seria um bom fit para aquela playlist. É possível, por exemplo, que todas as primeiras 10 músicas selecionadas para uma playlist fossem relevantes e que, por algum motivo, não compunham a playlist no momento da obtenção dos dados.

Neste caso, a predição do modelo seria uma boa predição, mas as métricas disponíveis para a avaliação não seriam capazes de perceber isso.

4. Conclusão

Com o que foi apresentado, é possível tirar algumas conclusões. Primeiramente, este é um problema bastante relevante que demanda soluções fora do que é comum para o problema de sistemas de recomendação.

Segundo, a estrutura autoencoder foi capaz de representar melhor o conjunto das músicas presentes numa playlist. Isso se dá, provavelmente, pelo fato de a estrutura não necessitar de outras playlists para ser representada num espaço vetorial.

Outro ponto a ser colocado é a quantidade de músicas e playlists presentes no subconjunto. Pode ser que as músicas excluídas no pré processamento apareçam mais no restante do dataset e, por um acaso, tiveram de ser excluídas, por exemplo.

Conclui-se, por fim, que problemas de recomendação em que o universo é muito maior do que o que um usuário é capaz de conhecer faz com que sejam necessárias outras estratégias de recomendação além de modelos de aprendizado de máquina.

REFERÊNCIAS

CHEN, C.W., LAMERE, P., SCHEDL, M., ZAMANI, H. **RecSys Challenge 2018:** Automatic Music Playlist Continuation. **ACM Conference on Recommender Systems (RecSys '18)**, outubro de 2018.

YANG, H., JEONG, Y., CHOI, M., LEE, J. **MMCF**: Multimodal Collaborative Filtering for Automatic Playlist Continuation. **Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18)**, out. de 2018.

ZAMANI, H., SCHEDL, M., LAMERE, P., CHEN, C.W. An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. **Association for Computing Machinery**, set. de 2019.