Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística
Programa de Especialização em *Data Science* e *Big Data*

Rafaela Souza Pinter

Continuação de playlists musicais utilizando sistema de recomendação baseado em feedback implícito

Curitiba
2022

Rafaela Souza Pinter

Continuação de playlists musicais utilizando sistema de recomendação baseado em feedback implícito

Monografia apresentada ao Programa de Especialização em *Data Science* e *Big Data* da Setor de Ciências Exatas, Universidade Federal do Paraná como requi-sito parcial para a obtenção do grau de espe-cialista.

Orientador: Prof.Dr. Walmes Marques Zeviani

Curitiba
2022

Continuação de playlists musicais utilizando sistema de recomendação baseado em feedback implícito

Criação de interface interativa para continuação de playlists personaliadas

Rafaela Souza Pinter[1]

[1]Engenheira de Bioprocessos e Biotecnologista*

Sistemas de recomendação são ferramentas robustas utilizadas para auxiliar clientes ou usuários a encontrarem itens relevantes para si. Com o advento de plataforma de *streaming* de músicas, o modo de se consumir músicas em playlist se popularizou, o que gerou um novo desafio de aplicar sistemas de recomendação para a continuação de playlists. O conjunto de dados do desafio Million Playlist Dataset do Spotify foi utilizado neste trabalho para avaliar diferentes implementações de sistemas de recomendação de feedback implícito. Além disso, foi construída uma interface web interativa utilizando o Plotly Dash para teste das recomendações para playlists personalizadas, com possibilidade de exportar a playlist continuada para a conta do Spotify do usuário. Foram testadas bibliotecas Spotlight, Implicit e LightFM, na linguagem Python, e se observou que a biblioteca Implicit gerou bons resultados nos menores tempos de execução. Utilizando a biblioteca Implicit, foram feitos estudos dos hiperparâmetros nas combinações de 5, 10, 100, 200 e 500 fatores, 5, 10, 20 iterações e 0,1 e 1 para o fator de regularização. O objetivo era identificar uma combinação de hiperparâmetros que executasse o mais rápido possível e gerasse boas recomendações. Os melhores resultados foram obtidos com a utilização de 500 fatores, porém a combinação de 200 fatores, 10 iterações e fator de regularização igual a 1 gerou resultados satisfatórios em menor tempo de execução. Como próximos passos, será compilada uma submissão para o desafio do Spotify e poderá ser estudada a aplicação de algoritmo de similaridade entre a playlist customizada e a base de dados para ser possível a utilização de modelo pré-treinado na aplicação, otimizando seu tempo de execução.

Palavras-chave: sistema de recomendação, filtro colaborativo, *features* implícito

Recommendation systems are robust tools used to help customers or users find items relevant to them. With the advent of music streaming platform, the way of consuming music in playlist has become popular, which has created a new challenge of applying recommender systems for the continuation of playlists. The challenge Million Playlist Dataset from Spotify was used in this work to evaluate different implementations of implicit feedback recommender systems. In addition, an interactive web interface was built using Plotly Dash for testing the recommendations for custom playlists, with the possibility of exporting the continued playlist to the user's Spotify account. Spotlight, Implicit and LightFM libraries were tested in the Python language, and it was observed that the Implicit library generated good results in the shortest runtimes. Using the Implicit library, studies of the hyperparameters were performed on combinations of 5, 10, 100, 200, and 500 factors, 5, 10, 20 iterations, and 0.1 and 1 for the regularization factor. The goal was to identify a combination of hyperparameters that would run as fast as possible and generate good recommendations. The best results were obtained using 500 factors, but the combination of 200 factors, 10 iterations, and a smoothing factor equal to 1 generated satisfactory results in less execution time. As next steps, a submission will be compiled for Spotify's challenge, and the application of a similarity algorithm between the customized playlist and the database can be studied to be able to use a pre-trained model in the application, optimizing its execution time.

Keywords: recommender systems, collaborative filtering, implicit feedback

1. Introdução

Em um mundo onde temos empresas oferecendo catálogos de itens na ordem de grandeza de milhões, como a Amazon, Spotify, Medium, temos cada vez mais pos-

*rafaelaspinter@gmail.com

sibilidades de escolha. Por um lado, a internet nos traz a liberdade e a possibilidade de contato com produtos que possivelmente não teríamos se em gerações passadas. Por outro, precisamos dispor cada vez de mais tempo para encontrar o que buscamos. Barry Schwartz, autor do livro *The Paradox of Choice*, diz justamente que quanto mais opções tivermos, menos satisfeitos com nossa escolha estaremos.

Dispor de muitas opções de produtos também gera o fenômeno de *Long Tail*: poucos produtos são muito populares, enquanto muitos produtos são pouco populares [1]. E, consequentemente, os produtos mais populares acabam sendo mais vendidos. Em lojas físicas, temos menos produtos disponíveis (ao menos no estoque local da loja) e vendedores que conhecem estes produtos, que conversam com os clientes, entendem suas necessidades. Estes vendedores trazem as opções cuidadosamente selecionadas que irão satisfazer a necessidade deste cliente. Por outro lado, no mundo web, consumidores entram em contato com um número demasiadamente grande de produtos e precisam explorar solitariamente catálogos de produtos que podem chegar a milhares a milhões. Assim, um importante desafio é criado para ajudar os consumidores no processo de encontrar, escolher e comprar os produtos mais relevantes para si [1]. Visando solucionar este tipo de problema, sistemas de recomendação ganharam importância no cenário de ciência de dados, principalmente com o desafio *Netflix Prize* [2]. Este desafio foi lançado em 2006 e consistia de um dataset com 100 milhões avaliações de filmes por usuários anonimizados (que posteriormente foram individualmente identificados, como descrito em [3] com o prêmio de 1 milhão de dólares para a pessoa ou equipe que aumentasse a acurácia do então sistema de recomendação da plataforma [4].

Em linhas gerais, sistemas de recomendação são ferramentas utilizadas para sugerir aos usuários seus itens de maior interesse. Por "itens"podemos entender: produtos para consumir, músicas para ouvir, artigos para ler, filmes para assistir. Existem diferentes abordagens para serem seguidas que implicam em diferentes dados e metadados utilizados no modelo, porém os mais comuns são algoritmos baseados em filtro colaborativo, em filtro de conteúdo e algoritmos híbridos [5].

Nos algoritmos de filtro colaborativo, entendemos puramente como os produtos são normalmente combinados ou avaliados pelos usuários, achando similaridade e padrões para realizar novas recomendações. Usualmente, a similaridade entre usuários é calculada pelo modo com que eles interagem com os itens [6], pois considera-se que usuários que interagem com os mesmos produtos de forma similar são usuários parecidos. Nesta abordagem, não há necessidade de se utilizar informações complementares sobre os itens ou quem são os usuários, o que traz a vantagem de ser uma técnica passível de ser aplicada em diferentes domínios [5]. Neste tipo de algoritmo, problemas de esparsidade são frequentemente observados quando aplicados em bases de dados de grandes volumes [6]. É muito comum organizar os dados em matrizes onde cada linha representa um usuário, e cada coluna representa um item disponível. Cada célula desta matriz é então preenchida com a interação do usuário com aquele respectivo item. Estas interações podem ser de *features* explícito ou *features* implícito.

Em *features* explícito, temos informações do que o usuário gosta e do que não gosta na forma de uma nota numérica dada para um produto. Por exemplo, número de estrelas, avaliação positiva ou negativa de um item. Já em *features* implícito, somente temos informação do que o usuário já interagiu, mas não sabemos necessariamente se ele gostou ou não do item. Por exemplo, podemos ter a informação de produtos comprados pelo usuário, filmes assistidos, músicas ouvidas ou itens visualizados. Porém, em nenhuma destas interações temos um *features* quantitativo sobre como foi a experiência do usuário com determinado item. Sabemos que houve consumo, mas não sabemos se foi agradável ou desagradável. Há alguns casos onde podemos ter mais confiança sobre o que o usuário gosta. Se um usuário ouve muitas vezes a mesma música, isso é um forte indicativo de preferência. Se um cliente compra diversos produtos de uma mesma marca, também pode-se esperar que haja uma preferência. Contudo, em todos estes casos, haverá, em geral, um número demasiadamente maior de itens que o usuário não viu, não conhece e não consumiu do que o contrário. Além disso, quando há inclusão de novos itens no catálogo ou de novos clientes, a abordagem de filtro colaborativo pode sofrer do que se chama de *cold start*. O *cold start* acontece quando há poucas informações de um usuário ou de um item na base de dados, gerando um grande problema, dado que justamente estas interações são as modeladas para realizarem as recomendações [6].

A abordagem de recomendação utilizando algoritmos de filtro baseado em conteúdo sofre menos com problemas de *cold start*. Esta abordagem visa recomen-

dar itens de características similares aos usuários. Características, estas, dadas por *features* específicas de cada item ou usuário. Por exemplo, no caso de músicas, pode-se levar em conta *features* como o gênero musical, ano de lançamento, artista, álbum, outras características específicas das músicas. Também é possível optar pela abordagem que utiliza características dos próprios usuários (como gênero, país e idade) para traçar o perfil de clientes que possuem interesses comuns. Visando mitigar limitações de ambas as abordagens anteriores, os sistemas de recomendação híbridos utilizam técnicas de filtro colaborativo para identificar interações comuns de usuários e itens, e de filtro de conteúdo para incluir *features* que os descrevam [5].

No cenário musical, o modo de consumo de música mudou drasticamente nos últimos anos com a crescente distribuição e disponibilização de músicas em plataformas de *streaming* como a Spotify, Pandora, Apple, Amazon, YouTube e Deezer [7]. De acordo com a pesquisa publicada pela Digital Musical Alliance em 2018, 54% dos consumidores dizem que playlists estão substituindo os álbuns nos seus hábitos de consumo de música. De fato, em 2018, mais de 4 bilhões de playlists já haviam sido criadas no Spotify. Com esta mudança de hábito, um novo desafio de recomendação de músicas surgiu: a continuação de playlists [9]. Dada uma playlist manualmente criada por um usuário, o objetivo do sistema de recomendação é acrescentar n músicas que estejam em harmonia com as demais. Esta pode ser uma tarefa não trivial, uma vez que músicas de anos, estilos, gêneros, países diferentes podem estar agrupadas em uma única playlist. Ainda, playlists podem conter um número pequeno de músicas, dificultando sua continuação e exemplificando uma situação de *cold start*.

Muitos sistemas de recomendação genéricos já estão disponíveis com implementações em Python, como por exemplo, as bibliotecas LightFM, Implicit, Surprise e LibRec, e podem ser utilizados para a criação de sistemas de recomendação de músicas [7]. Há também implementações que podem ser utilizadas para estudos de algoritmos baseados em vizinhança para exploração de *features* de músicas e playlist, como a Annoy, NMSLIB e faiss[7].

Logo, o objetivo deste trabalho é estudar sistemas de recomendação para continuação de playlists e criar uma interface interativa onde o usuário possa testar diferentes configurações de sistemas de recomendação. Através do estudo da base de dados disponível, foi possível o entendimento acerca de qual abordagem de recomendação utilizar. Uma vez definida, foi realizada uma pesquisa de bibliotecas que implementassem tal abordagem, possibilitando testes e avaliações da implementação mais eficiente, *i.e.*: baixo tempo de execução e boa qualidade de recomendações.

2. Revisão bibliográfica

O algoritmo de *Alternating Least Squares*, ALS, Mínimos Quadrados Alternados – descrito em Hu *et al.* 2008 [11], é baseado em matriz de fatoração. Ele avalia interações de usuários (u, v) e itens (i, v). Neste trabalho, as playlists foram consideradas como usuários e as músicas, como itens. Estas interações são mais comumente chamadas de observações, e são denotadas por r_{ui}. No caso de *feedbacks* explícitos, estas observações podem ser, por exemplo, a nota de 1 a 5 para o respectivo item. No caso de *features* implícito, podemos ter, por exemplo, o número de vezes que um produto foi comprado, o tempo total assistido de um filme, o número de vezes que uma música foi reproduzida, entre outros parâmetros.

Como dito anteriormente, em *feedback* implícito, não é possível ter informações do que o usuário gosta ou desgosta. Hu *et al.* 2008 [11] também introduz uma variável binária que formaliza a noção de preferência, p_{ui}, que representaria a preferência do usuário u pelo item i. Esta variável é definida por:

$$\mathrm{p}_{ui} = \begin{cases} 1 \; r_{ui} > 0 \\ 0 \; r_{ui} = 0 \end{cases}$$

Nos casos de *feedback* implícito, onde as observações r_{ui} são, por exemplo, o número de vezes que uma música foi reproduzida, usuários podem ouvir uma música porque gostaram dela, outros ainda podem ter músicas escolhidas por terceiros que não necessariamente sejam de sua predileção. Para lidar com estas situações, Hu *et al.* 2008 [11] introduz a noção de confiança que temos na observação. Em geral, com o aumento nos valores observados em r_{ui}, temos mais confiança c_{ui} de que o usuário definitivamente gosta daquele item. Logo, Hu *et al.* 2008 [11] define uma escolha de c_{ui} como:

$$\mathrm{c}_{ui} = 1 + \alpha r_{ui}$$

O grande desafio do algoritmo de ALS é encontrar os vetores de características x_u para usuário e y_i para o item onde a preferência p_{ui} será dada pelo produto interno $p_{ui} = x^T * u * y_i$. Extrapolando para o conjunto de dados completo, o objetivo do algoritmo ALS é fatorar a matriz de observações R (onde linhas representam o conjunto de playlists e as colunas representam o

conjunto de músicas) em uma matriz de características do usuário (ou playlist) X e matriz de características dos itens (músicas) Y, de forma que $R = XY$.

Hu *et al.* 2008 [11] utiliza como função de custo:

$min_{x*,y*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$

onde:

$\lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$

é o fator de regularização que previne o modelo de apresentar *overfitting* para os dados. É sugerido que λ seja definido por validação cruzada. As matrizes X e Y são definidas pela minimização desta função de custo. Como o nome sugere, Hu *et al.* 2008 [11] implementam o método de mínimos quadrados para realizar esta otimização. O termo *"alternating"* vem do fato que ambas as matrizes X e Y são alternadas na minimização do erro. Ou seja:

1. As matrizes X e Y são inicializadas;
2. A matriz X é fixada e Y é otimizada para se obter a matriz original R;
3. A matriz Y é fixada e X é otimizada para se obter a matriz original R.

Estes passos são repetidos por n iterações.

Para a realização das recomendações, a matriz R é calculada, dada por $R = XY$, onde para cada usuário (playlist), recomenda-se os itens (músicas) com maior escore calculado.

3. Métodos

3.1. Base de dados

Os dados utilizados neste projeto foram obtidos a partir do desafio aberto e contínuo do Spotify Million Playlist Dataset [10], organizado pelo Spotify, pela University of Massachusetts e pela Johannes Kepler University. Este desafio, lançado em 2020, é uma continuação de um primeiro desafio que aconteceu entre janeiro e julho de 2018. A principal tarefa de ambos é a criação de sistema de recomendação para a continuação de playlists. Assim, dada uma playlist arbitrária, os participantes devem recomendar 500 novas músicas com características similares às da playlist original. No desafio original de 2018, houve registro de 1791 participantes, sendo 1430 com filiação acadêmica e 361 vindos da indústria. No total, 410 times foram compostos por estes participantes.

A base de dados disponibilizada pelo Spotify consiste de uma amostra de 1 milhão de playlists reais criadas entre janeiro de 2010 e novembro de 2017, de um total de 4 bilhões de playlists públicas do Spotify. Os dados foram disponibilizados particionados em 1000 arquivos contendo cada um 1000 playlists. Cada um destes arquivos, disponibilizados em formato JSON, consiste em informações da playlist e informações das músicas que a compõem, como no código na Figura 1.


```
{
    "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"
    },
    // 9 tracks omitted
    {
        "pos": 11,
        "artist_name": "Mo' Horizons",
        "track_uri":
            "spotify:track:7iwx00eBzeSSSy6xfESyWN",
        "artist_uri":
            "spotify:artist:3tuX54dqgS8LsGUvNzgrpP",
        "track_name": "Fever",
        "album_uri":
            "spotify:album:2Fg1t2tyOSGWkVYHlFfXVf",
        "duration_ms": 364320,
        "album_name": "Come Touch The Sun"
    }
    ],
}
```


Figura 1: JSON inicial disponibilizado na Spotify Million Playlist Dataset

3.2. Bibliotecas testadas

Para implementação do sistema de recomendação, foram testadas 3 bibliotecas implementadas na linguagem de programação Python: Implicit, LightFM e Spotlight.

Implicit é uma biblioteca *open source* com implementações de filtro colaborativo para conjunto de dados de *feedback* implícito. Conta com algoritmos de Alternating Least Squares, Bayesian Personalized Ranking e modelo Item-Item Nearest Neighbors que utilizam métodos de Cosseno, TFIDF e BM25 como cálculo de distância. Todos os métodos são implementados em Cython e OpenMP, onde o treino do modelo é feito em paralelo na CPU. Há também a possibilidade do modelo ser treinado em GPU, caso disponível. Para este projeto, foi testado a implementação do algoritmo de Alternating Least Squares. A implementação da biblioteca Implicit utiliza como base a simples solução de recomendação com base em matriz de fatorização descrita por Hu *et al.* 2008 [11]. Em Takács *et al.* 2011 [13], porém, temos a sugestão de uma alteração na função de otimização que passa de mínimos quadrados para gradiente conjugado. Essa alteração aumenta a velocidade do algoritmo.

LightFM é uma biblioteca de Python com implementação de uma série de algoritmos populares de sistemas de recomendação, tanto para *features* implícito, quanto para explícito. A grande vantagem desta biblioteca é a implementação de um modelo híbrido, utilizando técnica de matriz de interação e a aplicação de metadados sobre músicas e playlists. Possui quatro implementações de funções de custo que podem ser utilizadas conforme as diferentes bases de dados utilizadas. Para este projeto, foi testada a implementação para *feedback* implícito com utilização da função de perda Ranking Bayesiano Personalizado [14].

A biblioteca Spotlight implementa diversos algoritmos para a construção de sistemas de recomendação, contendo uma implementação de matriz de fatoração descrita por Yehuda *et al.* 2009 [8] utilizando a biblioteca PyTorch, utilizada para o teste em questão. Foi utilizada a mesma função de perda de Ranking Bayesiano Personalizado.

3.3. Ferramentas para criação da interface do usuário

O Plotly Dash é um framework open source para a criação de interfaces de visualização. Está disponível como biblioteca para Python, R e Julia. Neste projeto, foi utilizado para a criação de uma aplicação web interativa para o teste de continuação de playlist, possibilitando ao usuário utilizar playlist personalizada, customizar o algoritmo de recomendação e visualizar as diferentes qualidades das recomendações. Ao treinar um modelo, pode-se visualizar diversos gráficos que exploram os metadados das músicas, comparando as músicas originais da playlist com as recomendadas. A recuperação destes metadados das músicas foi feita utilizando a web API do Spotify.

Visando possibilitar o desenvolvimento de aplicações, o Spotify possui uma API web baseada nos princípios REST. Seus endpoints retornam JSON com metadados de artistas, álbuns e músicas. Assim, utilizando os identificadores individuais *ids* das músicas já disponibilizados pela Million Playlist Dataset, é possível fazer as requisições de cada música da playlist. As respostas da API para os metadados das músicas envolvem diferentes métricas. Para comparação neste projeto, foram utilizadas:

- *acousticness*: um nível de confiança entre 0.0 e 1.0 sobre caso a música seja acústica. Valores iguais a 1.0 representam alta confiança que a música é acústica.

- *danceability*: descreve quão adequada a música é para se dançar, baseado em uma combinação de ritmo em bpm (batidas por minuto), estabilidade do ritmo, força da batida e regularidade em geral.

- *energy*: métrica entre 0.0 e 1.0 que representa a métrica perceptual da intensidade da música.

- *instrumentalness*: métrica entre 0.0 e 1.0 que prediz caso a música não contenha vocal.

- *liveness*: detecta a presença de audiência na música, com escore variando de 0.0 a 1.0.

- *speechness*: detecta a presença de palavras faladas na música, diferenciando áudio com somente palavras faladas (poesia, audiobooks), de áudios com música e fala, e áudios com somente música.

- *valence*: medida de 0.0 a 1.0 que indica a positividade da música. Valores próximos de 1.0 indicam músicas mais positivas (alegres, eufóricas), enquanto valores próximos de 0.0 indicam músicas negativas (tristes, depressivas).

A API web também fornece ferramentas para acesso de dados de usuários, como músicas salvas, playlists criadas e seguidas. Além disso, com a autenticação do usuário, torna-se possível a utilização das suas playlists customizadas para sua continuação. O usuário pode fazer *login* na interface com sua conta Spotify, obter recomendações de novas músicas para uma playlist de sua escolha e comparar estas diferentes recomendações

utilizando diferentes parametrizações do algoritmo. Caso deseje, poderá exportar a playlist continuada de volta para sua conta.

Desta forma, a interface foi utilizada para comparar recomendações de uma playlist teste e playlist simulando uma situação de *cold start*.

3.4. Teste de hiperparâmetros

Este teste visa determinar bons hiperparâmetros para a execução do código na sua aplicação. O objetivo é ter uma recomendação com um balanço de boa qualidade, com músicas que façam sentido naquela playlist, que contenha com um fator de novidade, e, principalmente, baixo tempo de execução do modelo. Este balanço de qualidade e velocidade de execução consistem nas configurações ideais dos hiperparâmetros padrão para a aplicação web.

O teste consistiu na aplicação de um *grid search* para criar todas as diferentes combinações de hiperparâmetros selecionados para o experimento de análise na biblioteca Implicit, que implementa o algoritmo de Alternating Least Squares. Como a base de dados utilizada neste projeto não contém informações sobre o número de reproduções de cada música, considerou-se a observação r_{ui} como um valor *booleano* representando a presença ou ausência da música na playlist. Ou seja, caso uma música i esteja na playlist u, seu valor r_{ui} será 1. Consideramos $r_{ui} = 0$ para todas as outras músicas do conjunto completo de músicas que não estão presentes na playlist u. Logo, como r_{ui} sempre será um valor igual a 0 ou 1, temos que $p_{ui} = r_{ui}$.

O valor de α, por mais que seja um dos hiperparâmetros possíveis de serem otimizados, não foi utilizado nos testes. O α representa um fator multiplicador da observação (r_{ui}) para o cálculo da confiança c_{ui}. Este passo é importante quando queremos dar mais peso para observações de alto valor. Caso tivéssemos também o número de vezes que uma música foi reproduzida, poderíamos testar diferentes valores de α para visualizar diferentes pesos. Contudo, como os valores utilizados na matriz são binários, optou-se por manter um valor de alfa fixo em 1 neste trabalho. Assim, este fator alfa não foi otimizado para simplificar a avaliação.

Para o número de fatores, foram utilizados os valores de 5, 10, 100, 200 e 500. Para a regularização, 0,1 e 1. As iterações variaram entre 5, 10 e 20. Uma mesma playlist foi utilizada em todos os testes, e pode ser encontrada na Tabela 2.

A avaliação dos resultados de músicas recomendadas foi feita de três maneiras: (a) qualitativa-manual; (b) comparativa; e (c) por comparação de *features* musicais obtidas pela API do Spotify. Na avaliação qualitativa-manual, as recomendações foram observadas e interpretadas uma a uma a fim de se entender os padrões de popularidade e de similaridade com a playlist original, a variação da ordem das recomendações e os elementos de novidade (músicas que o usuário não conhecia antes). Na avaliação comparativa, as playlists foram comparadas para que a similaridade fosse observada em termos de porcentagem de *ids* de músicas idênticas presentes nas recomendações. Por fim, as músicas recomendadas e originais foram comparadas em termos de suas *features*.

```
{
    "5": [
            "7vqa3sDmtEaVJ2gcvxtRID",
            "23EOmJivOZ88WJPUbIPjh6",
            // 9 tracks omitted
            "7iwx00eBzeSSSy6xfESyWN",
        ],
}
```

Figura 2: JSON processado

4. Resultados

A base de dados trabalhada neste projeto foi constituída de 1.000.000 (Tabela 1) playlists coletadas entre 2010 e 2017, sendo inteiramente coletadas de usuários dos Estados Unidos da América. Destes, 45% são do gênero masculino, 54% do gênero feminino, 0,5% se identificam como não-binário e 0,5% não especificaram gênero. Os jovens estão entre os mais presentes nos criadores de playlists: 43% dos usuários estão na faixa de 18 a 24 anos, seguido por 31% de 25 a 34 anos, e 10% de 13 a 17 anos. Usuários +35 anos representam 16% dos usuários na base de dados.

A base de dados precisou passar por transformações para que pudesse ser utilizada no modelo. Das informações disponíveis originalmente, optou-se por criar uma lista de *track_uri* contidos em cada playlist. Assim, o tamanho do arquivo utilizado para as análises foi reduzido de 33 gigabytes para 2,28 gigabytes, otimizando a utilização de memória do computador. Ao final, a base utilizada continha as seguintes informações descritas na Figura 2.

Métrica	Valor
Número de playlists	1.000.00
Número total de músicas	66.346.428
Número de músicas únicas	2.262.292
Número de álbuns únicos	734.684
Número de artistas únicos	295.860
Número médio de músicas por playlist	66,34

Tabela 1: Métricas da base de dados

4.1. Comparação de bibliotecas

4.1.1. Spotlight

A biblioteca Spotlight foi a primeira a ser testada e a se obter sucesso com a construção de um sistema de recomendação para continuação de playlist. Contudo, o tempo necessário para o treinamento dos modelos foi demasiadamente grande, dificultando a utilização da base de dados completa. Utilizando apenas uma amostra de 2000 playlists, o que representa 0,2% da base completa, o algoritmo levou 97 minutos para treinar. Ademais, devido ao pequeno número de playlists e músicas utilizadas no treino, as recomendações nas continuações de playlists não foram satisfatórias e não apresentavam semelhança com as playlists originais. Assim foi necessária a substituição da biblioteca por outra de maior eficiência.

4.1.2. Implicit

Para a implementação da segunda tentativa, foi utilizada a biblioteca Implicit. Ao contrário da primeira, esta biblioteca apresentou baixos tempos de treino do modelo utilizando sua base completa. Pode-se atribuir os baixos tempos de execução da playlist a dois pontos principais: (a) foram utilizadas matrizes esparsas para o treino do modelo, cujas estruturas de dados são majoritariamente elementos zerados nas entradas; e (b) a otimização utilizada pela biblioteca é baseada no método do gradiente conjugado.

Mesmo antes de realizar o estudo dos hiperparâmetros do modelo, já foi possível obter resultados satisfatórios utilizando a combinação de 100 fatores, 5 iterações e fator de regularização igual a 1. Nesta configuração, por exemplo, temos uma playlist de rock com as músicas descritas na Tabela 2, e as recomendações para esta playlist descritas na Tabela 3.

De um ponto de vista qualitativo, as músicas de fato fazem sentido dentro do contexto da playlist original. Observa-se que as músicas recomendadas são populares e pertencentes ao gênero musical rock e soft rock. Utilizando as *features* específicas de cada música, podemos observar que, em média, as *features* das músicas apresentam similaridade entre si, como mostrado pela Figura 6.

Posição	Música, Artista
1	Ramble On, **Led Zeppelin**
2	Good Times Bad Times, **Led Zeppelin**
3	Travelling Riverside Blues, **Led Zeppelin**
4	Sultans Of Swing, **Dire Straits**
5	Walk Of Life, **Dire Straits**
6	Burnin' for You, **Blue Öyster Cult**
7	This Charming Man, **The Smiths**
8	Lovesong, **The Cure**
9	Bigmouth Strikes Again, **The Smiths**
10	My Sharona, **The Knack**
11	Ride 'Em On Down, **The Rolling Stones**
12	Who'll Stop The Rain, **Creedence Clearwater Revival**
13	Whiskey In The Jar, **Thin Lizzy**
14	London Calling, **The Clash**
15	Rock the Casbah, **The Clash**
16	She Sells Sanctuary, **The Cult**
17	Brown Eyed Girl, **Van Morrison**
18	Hit Me With Your Best Shot, **Pat Benatar**
19	Should I Stay or Should I Go, **The Clash**
20	The Letter, **The Box Tops**
21	Green Onions, **Booker T. & the M.G.'s**
22	Rain, **The Cult**
23	Traveling Riverside Blues, **Eric Clapton**
24	Uptown Girl, **Billy Joel**
25	Build Me Up Buttercup, **The Foundations**
26	Jessie's Girl, **Rick Springfield**

Tabela 2: Músicas da playlist de teste

Posição	Música, Artista
1	Take On Me, **a-ha**
2	Fortunate Son, **Creedence Clearwater Revival**
3	Come On Eileen, **Dexys Midnight Runners**
4	Don't You (Forget About Me), **Simple Minds**
5	Bad Moon Rising, **Creedence Clearwater Revival**
6	You Make My Dreams, **Daryl Hall & John Oates**
7	Everybody Wants To Rule The World, **Tears For Fears**
8	Have You Ever Seen The Rain?, **Creedence Clearwater Revival**
9	Gimme Shelter, **The Rolling Stones**
10	Tainted Love, **Soft Cell**

Tabela 3: Continuação da playlist de teste

Este resultado é interessante pois mesmo sem utilizar estas *features* na composição de dados para o treino do modelo, o algoritmo foi capaz de recomendar músicas com características sonoras semelhantes.

Esta mesma configuração foi utilizada para um teste de *cold start* em que a playlist de base utilizada possui poucas músicas, como mostrado na Tabela 4.

Avaliando as *features* das músicas recomendadas, observa-se que elas são extremamente diferentes das *features* da playlist de *cold start*, como mostrado pela Tabela 6 e Figura 4.

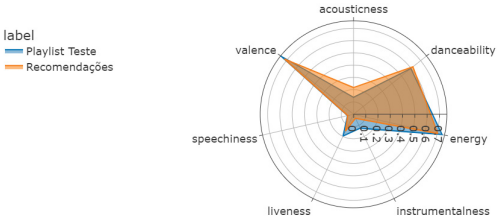


Figura 3: Comparação de *escores* médios entre Playlist Teste e recomendações de continuação utilizando 100 Fatores, 5 Iterações e Regularização igual a 1

Posição	Música, Artista
1	Jamie All Over, **Mayday Parade**
2	Black Butterflies and Déjà Vu, **The Maine**

Tabela 4: Playlist de teste cold start

Posição	Música, Artista
1	Te Metiste, **Ariel Camacho y Los Plebes Del Rancho**
2	Solo Con Verte, **Banda Sinaloense MS de Sergio Lizárraga**
3	Me Vas a Extrañar, **Banda Sinaloense MS de Sergio Lizárraga**
4	Adiós Amor, **Christian Nodal**
5	Después de Ti, ¿Quién?, **La Adictiva Banda San José de Mesillas**
6	Hablemos, **Ariel Camacho y Los Plebes Del Rancho**
7	No Lo Hice Bien, **Los Plebes del Rancho de Ariel Camacho**
8	Ya Te Perdí La Fe, **La Arrolladora Banda El Limón**

Tabela 5: Continuação de playlist teste *cold start*

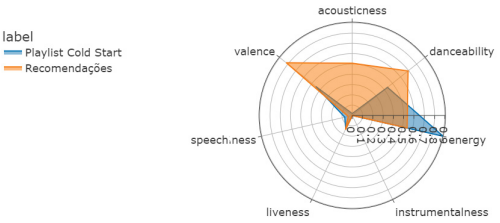


Figura 4: Comparação de *escores* médios entre Playlist Teste e recomendações de continuação utilizando 100 Fatores, 5 Iterações e Regularização igual a 1

Porém, incluindo apenas mais uma música na Playlist de *cold start* – a música Misery Business, Paramore –, o cenário muda completamente. As músicas recomendadas passam a ser de artistas notavelmente similares, como mostrado pela Tabela 6 e Figura 5.

Observando o número de playlist em que cada música se encontra Black Butterflies and Déjà Vu, de The Maine, está em 190 playlists; Jamie All Over, de Mayday Parade, em 1565; e Misery Business, de Paramore, em 7323. Possivelmente há um viés de músicas mais populares, onde estas tendem a ser mais importantes

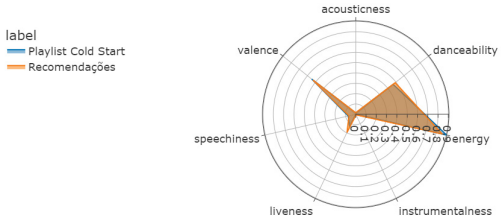


Figura 5: Comparação de *escores* médios entre Playlist de Cold Start complementada com Misery Business, Paramore e recomendações de continuação utilizando 100 Fatores, 5 Iterações e Regularização igual a 1

Posição	Música, Artista
1	Sugar, We're Goin Down, **Fall Out Boy**
2	I Write Sins Not Tragedies, **Panic! At The Disco**
3	Thnks fr th Mmrs, **Fall Out Boy**
4	Welcome to the Black Parade, **My Chemical Romance**
5	Ocean Avenue, **Yellowcard**
6	Dance, Dance, **Fall Out Boy**
7	Teenagers, **My Chemical Romance**
8	All The Small Things, **blink-182**
9	Dirty Little Secret, **The All-American Rejects**

Tabela 6: Continuação da playlist de teste *cold start* com inclusão de Misery Business, Paramore

para a geração das recomendações, uma vez que estão presentes em mais playlists.

4.1.3. LightFM

A biblioteca LightFM implementa sistema de recomendação híbrido utilizando matriz de fatorização que possibilita a adição de *features* das músicas. Esta biblioteca foi testada para avaliar sua eficiência e qualidade de recomendação. Como mencionado anteriormente, sistemas de recomendação híbridos têm vantagens em situações de *cold start* por utilizarem *features* das músicas ou playlists. A utilização desta biblioteca obteve sucesso em testes iniciais. Nestes primeiros testes, foi utilizado somente o algoritmo de filtro colaborativo para modelar a mesma base de dados utilizada para a biblioteca Implicit, que continha somente os *ids* das músicas presentes em cada playlist. Os primeiros testes mostraram que o algoritmo é capaz de realizar boas recomendações em baixos tempos de execução.

Há um grande potencial de atrelar metadados dos itens no sistema de recomendação, como mostrado por Kula, 2015[12], principalmente em casos de *cold start*. Nestes casos, temos poucas informações para fazermos recomendações personalizadas.

O Spotify tem uma API pública para consulta de artistas, álbuns e músicas, diretamente do Spotify Data Catalogue. Essa API foi utilizada para buscar os metadados das músicas presentes no dataset de treino, utilizando a requisição Get Track *features*. Porém, ao tentar executar as requisições para as 2.262.292 músicas únicas da base de dados, o limite de requisições foi atingido por volta das 10.000 primeiras. Nesta primeira tentativa, não foi utilizado um espaçamento de tempo entre as requisições para não sobrecarregar o servidor do Spotify. Porém, caso fosse optado pela utilização de um espaço de tempo entre as requisições de um segundo, por exemplo, não haveria tempo hábil para a execução do código. Considerando todas as músicas únicas, teríamos 2.262.292 s (aproximadamente, considerando que as demais operações não tenham tempo significativo para a estimação) de processamento. Este número equivale a mais de 26 dias de execução do código para se obter os metadados para as músicas do dataset. Infelizmente, este tempo de processamento é inviável para o tempo deste projeto.

Visando também encontrar outros conjuntos de dados públicos para buscar as *features* das músicas, foram analisados outros 19 arquivos CSV presentes na plataforma Kaggle. Destes arquivos, foram filtrados os que haviam campo de id das músicas, a fim de possibilitar um cruzamento das bases, totalizando 17 arquivos. Os arquivos foram concatenados e deduplicados a nível de id da música, totalizando 2.302.060 músicas únicas. Este conjunto de dados, então, foi comparado com o dataset de 1 milhão de playlists em uma operação de *inner join*, resultando na correspondência de apenas 314.570 *ids*. Número que, infelizmente, representa apenas 13,9% da base original, impossibilitando o uso do sistema de recomendação híbrido.

Logo, como não foi possível utilizar a API disponibilizada pelo Spotify e nem o conjunto de dados do Kaggle, não foi possível utilizar a principal vantagem da biblioteca, isto é, a utilização dos metadados das músicas em um sistema de recomendação híbrido.

Em termos de comparação, as recomendações feitas pela biblioteca LightFM foram similares às apresentadas pela biblioteca Implicit. Porém, o tempo de execução do treino do modelo para a biblioteca LightFM foi maior para se obter tais resultados. Foi necessário 126 min para treinar o modelo com 100 fatores e 5 iterações, e 22 minutos com 10 fatores e 5 iterações. Assim, com o objetivo de criar a interface de recomendação, foi descartada a utilização desta biblioteca.

Devido à sua simplicidade, melhores resultados e menores tempos de treino do modelo, a biblioteca Implicit foi a escolhida para a aplicação e para o estudo dos melhores hiperparâmetros a serem utilizados na interface web. As telas criadas para a interface estão representadas no Anexo A, e o código fonte em `https://github.com/rafpinter/music-recommender-system`.

4.2. Teste de hiperparâmetros

Visando identificar hiperparâmetros adequados para serem utilizados na interface web, foram testadas todas as combinações de 5, 10, 100, 200 e 500 fatores, 0,1 e 1 para a regularização, e iterações variando entre 5, 10 e 20 utilizando a biblioteca Implicit. Desta forma, devido à natureza do algoritmo de matriz de fatorização que exige que a playlist do usuário esteja na base de treino, os resultados foram analisados não só em relação à qualidade dos resultados, como também em eficiência. O algoritmo ideal deve: (a) indicar boas recomendações; (b) ter fatores de novidade (ou seja, apresentar também músicas que o usuário não conheça); e, principalmente, (c) ter o menor tempo de execução possível para evitar desistências do usuário.

Em termos qualitativos de músicas recomendadas, não houve diferenças significativas entre os diferentes valores de regularização testados. Observa-se que quanto maior o número de fatores utilizados nos cálculos, o tempo de execução dos testes com regularização igual a 1 tendem a ser maiores.

Entre as diferentes iterações, podemos observar (Figura 7) que o tempo de execução do modelo tende a apresentar características lineares para os pontos testados. Contudo, em termos qualitativos de recomendações, observou-se pouca mudança das músicas recomendadas. Houve, sim, uma mudança de ordem das recomendações, porém estas músicas tendem a ser majoritariamente iguais entre os diferentes números de iterações dentro do mesmo número de fatores do modelo.

De fato, dentre as condições testadas no experimento, a alteração de fatores foi a que mais alterou qualitativamente as recomendações fornecidas pelo algoritmo, como mostrado pela Figura 8, que descreve a porcentagem de equidade entre as diferentes recomendações de músicas de cada condição testada. Vemos uma tendência de coloração próxima entre números próximos de fatores no teste. Por exemplo, recomendações feitas com algoritmo utilizando 5 fatores são mais similares com recomendações feitas com 10 fatores do que com 100, 200 ou 500. De forma similar, as recomendações

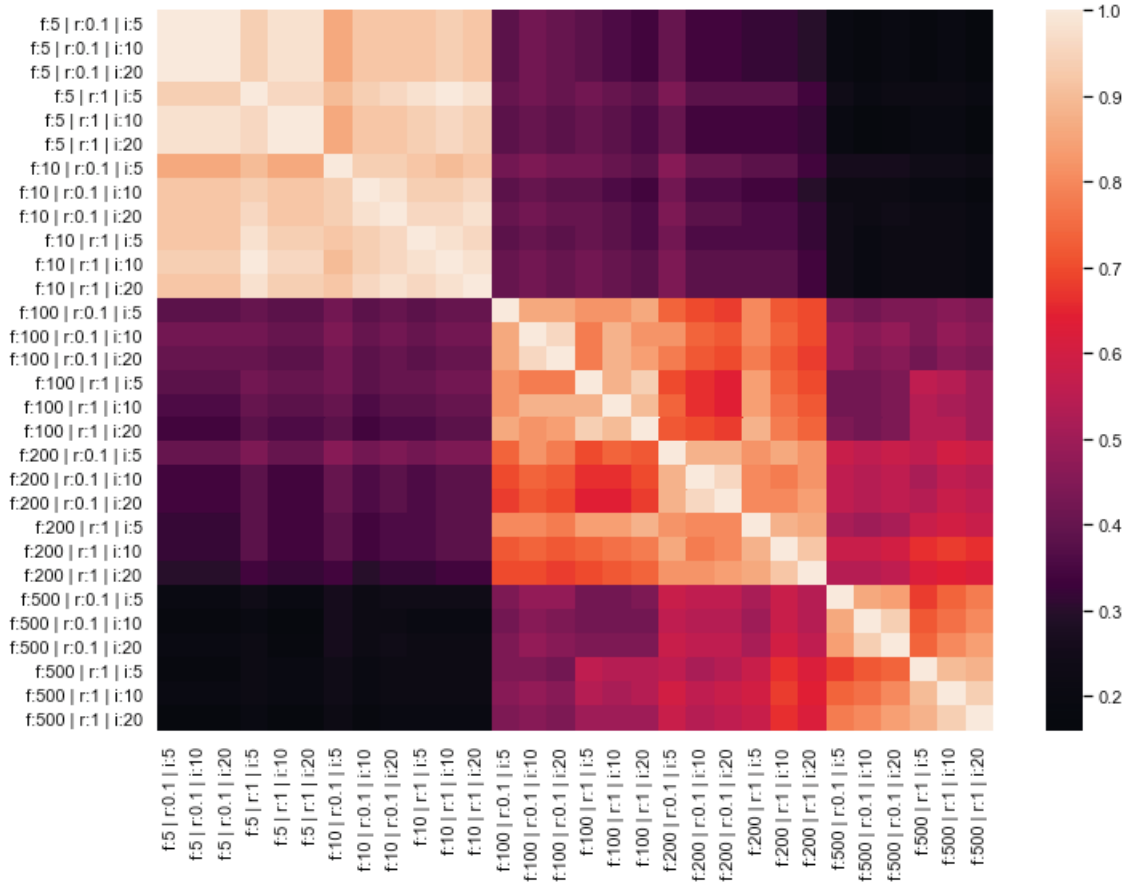


Figura 6: Similaridade entre playlists em configuração de treinos diferentes
f: Fatores r: Regularização i: Iterações

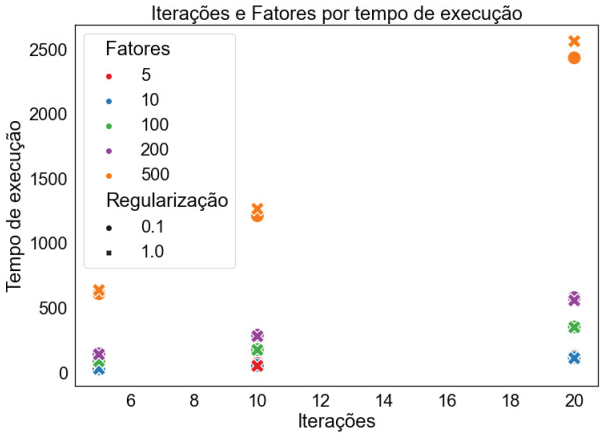


Figura 7: Tempo de execução do algoritmo em diferentes condições de treino

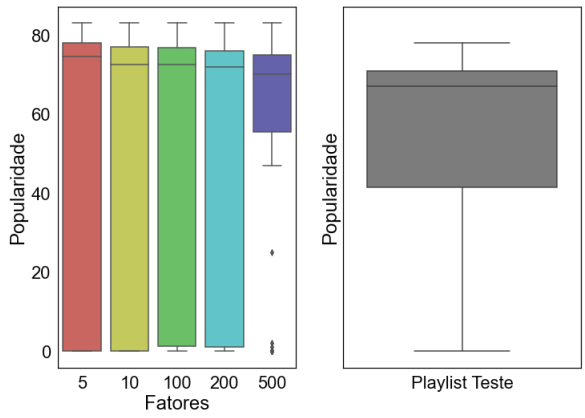


Figura 8: Comparação de *escores* de popularidade entre as recomendações e a playlist original

obtidas com 100 fatores são mais parecidas com as obtidas por 200 fatores do que com as demais.

Em geral, as recomendações utilizando 5 e 10 fatores produziram recomendações coerentes, porém não traziam elementos de novidade para a playlist, sendo majoritariamente músicas amplamente conhecidas. Como pode-se observar na Figura 8, os escores de po-

Fatores	Regularização	Iterações	% Correspondência
100	0.1	5	8.57%
100	0.1	10	8.57%
100	0.1	20	8.57%
100	1.0	5	8.57%
100	1.0	10	8.57%
100	1.0	20	8.57%
200	0.1	5	11.42%
200	0.1	10	11.42%
200	0.1	20	11.42%
200	1.0	5	8.57%
200	1.0	10	11.42%
200	1.0	20	14.28%
500	0.1	5	8.57%
500	0.1	10	5.71%
500	0.1	20	5.71%
500	1.0	5	8.57%
500	1.0	10	8.57%
500	1.0	20	5.71%

Tabela 7: Correspondências de recomendações pelo sistema de recomendação desenvolvido neste trabalho e recomendações fornecidas pelo Spotify nas diferentes configurações de hiperparâmetro

pularidade tendem a ser maiores quanto menor o número de fatores.

Já com 100 fatores, observa-se a recomendação de mais músicas menos populares, como mostra a Figura 7, porém foi somente a partir de 200 fatores que realmente se observou fatores de novidade para as recomendações. Como esperado, os testes utilizando 500 fatores se mostraram com as melhores recomendações, tendo uma diversidade maior de artistas e distribuição de popularidade das músicas muito semelhante à playlist testada.

Por fim, as recomendações de cada playlist foram comparadas com as recomendações fornecidas diretamente pelo Spotify utilizando a opção de "Criar playlist parecida"para a Playlist Teste. Observando a Tabela 7, onde temos o cálculo da porcentagem de correspondência das recomendações do Spotify com as top 50 recomendações calculadas, podemos notar que há uma semelhança a nível de músicas entre as recomendações. O melhor ponto de correspondência foi observado na configuração de 200 fatores, 20 iterações e fator de regulação igual a 1. Porém, deve-se notar que as recomendações do Spotify não possuem limitação de base de dados, como nas recomendações calculadas. Quando filtramos a playlist recomendada pelo Spotify para conter somente músicas presentes no conjunto de dados Million Playlist Dataset, este valor de correspondência sobe para 27,7%. O que indica que aproximadamente um terço das recomendações fornecidas Spotify estão presentes nas recomendações do sistema de recomendação aqui criado.

5. Conclusão

Com este projeto, foi possível analisar diferentes implementações de sistemas de recomendação para o desafio de continuação de playlists. Ao todo, três diferentes bibliotecas Python foram testadas: Spotlight, LightFM e Implicit. A biblioteca Spotlight produziu resultados insatisfatórios ao exigir longos períodos de processamento, inviabilizando, assim, sua utilização em uma interface web. Já a biblioteca LightFM produziu resultados satisfatórios, porém não foi possível utilizar seu sistema de recomendação híbrido devido à limitação das informações de *features* de todas as músicas da base, onde a limitação foi causada pelo tempo de processamento para a obtenção de todas as 2.262.292 músicas únicas na base de dados a partir de requisições na API do Spotify. Por fim, a biblioteca Implicit apresentou resultados satisfatórios e os menores tempos de execução, sendo adotada para a construção de uma interface interativa para que usuários possam criar suas próprias continuações, explorar o impacto dos diferentes hiperparâmetros no treino do modelo e exportar para o Spotify a playlist continuada. Dos hiperparâmetros disponíveis para serem otimizados pelo algoritmo de Alternating Least Squares, foram estudados o número de fatores, número de iterações e valor do termo de regularização. Ao final dos testes de hiperparâmetros, a aplicação web recebeu os valores de 200 para número de fatores, 20 para número de iterações e 1 para o valor do termo de regularização. Esta configuração de hiperparâmetros, por mais que não tenha sido otimizada pela aplicação de técnicas específicas, é capaz de gerar resultados satisfatórios em baixo tempo de execução.

Infelizmente, retreinar o modelo a cada vez que se deseja utilizar uma nova playlist é um fator limitante ao uso de valores altos para fatores e iterações do algoritmo, uma vez que o tempo de processamento cresce a valores impraticáveis para uma aplicação web. Contudo, seria possível calcular uma similaridade entre playlist que poderia ser aplicável para gerar as recomendações com base em outras playlist similares já presentes na base de treino [15]. Desta forma, um modelo já treinado poderia ser utilizado e servir para predições. Em contrapartida, a utilização de modelos treinados em produção pode sofrer o que se chama de *mudança de conceito*, onde o modelo se torna defasado pela atualização de novos dados. No exemplo trabalhado, novas músicas são constantemente criadas, popularizadas ou esquecidas. Desta forma, há introdução de uma nova camada de complexidade, onde seria

necessário acompanhar constantemente como as músicas são agrupadas, possibilitando sempre a melhor recomendação.

Como próximos passos, pode-se explorar a criação de um algoritmo de similaridade entre playlists para servir um modelo treinado com um maior número de fatores e de iterações. Além disso, o material estudado será organizado para a submissão no desafio Million Playlist Dataset Challenge disponibilizado pelo Spotify.

Agradecimentos

Gostaria de agradecer ao corpo docente da pós-graduação em Data Science & Big Data da Universidade Federal do Paraná pelos ensinamentos, especialmente ao professor Dr. Walmes Marques Zeviani pela orientação e à minha família pelo apoio e carinho durante esta jornada.

Referências

[1] H. Yin, B. Cui, J. Li, J. Yao, C. Chen, *Challenging the Long Tail Recommendation*, (CoRR), (2012).

[2] C. C. Aggarwal, *Recommender Systems, The Textbook*, (Springer), (2016).

[3] A. Narayanan, V. Shmatikov, *Robust De-anonymization of Large Datasets(How to Break Anonymity of the Netflix Prize Dataset)*, (arXiv), (2006).

[4] J. Bennett, S. Lanning, *The Netflix Prize*, (Netflix), (2009).

[5] L. J. Chew, S. C. Haw, S. Subramaniam, *Recommender System for Retail Domain: An Insight on Techniques and Evaluations*, (Association for Computing Machinery), (2020).

[6] F. Ricci, L. Rokach, B. Shapira, *Recommender Systems: Techniques, Applications, and Challenges*, (Recommender Systems Handbook), (2021).

[7] M. Schedl, P. Knees, B. McFee, D. Bogdanov, *Music Recommendation Sytems: Techniques, Use Cases, and Challenges*, (Recommender Systems Handbook), (2022).

[8] K. Yehuda, R. Bell, C. Volinsky, *Matrix factorization techniques for recommender systems*, (Computer 42.8), (2009)

[9] M. Volkovs, H. Rai, Z. Cheng, G. Wu, Y. Lu, S. Sanner, *Two-stage Model for Automatic Playlist Continuation at Scale*, (Proceedings of Proceedings of the ACM Recommender Systems Challenge 2018), (2018).

[10] C.W. Chen, P. Lamere, M. Schedl, and H. Zamanim (Recsys Challenge 2018: Automatic Music Playlist Continuation), (In Proceedings of the 12th ACM Conference on Recommender Systems, RecSys '18), (2018).

[11] Y. Hu, Y. Koren, C. Volinsky, *Collaborative Filtering for Implicit Feedback Datasets*, (Eighth IEEE International Conference on Data Mining), (2008).

[12] M. Kula, *Metadata Embeddings for User and Item Cold-start Recommendations*, (arXiv preprint arXiv:1507.08439), (2015).

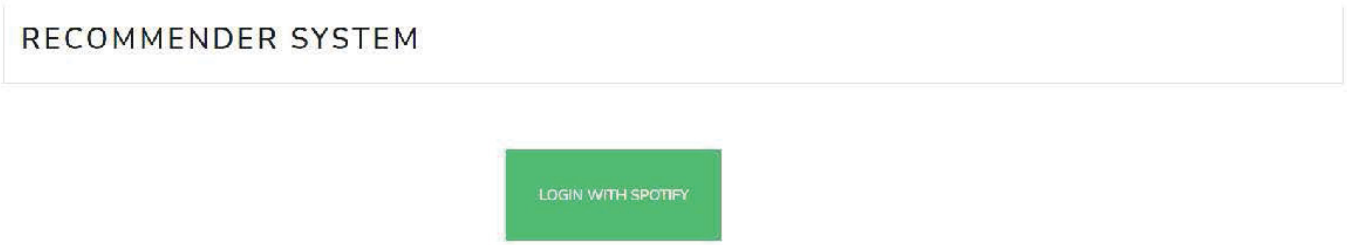
[13] G. Takács, I. Pilászy, D. Tikk, *Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering*, (RecSys '11), (2011).

[14] S. Rendle , *BPR: Bayesian personalized ranking from implicit feedback*, (AUAI Press), (2009).

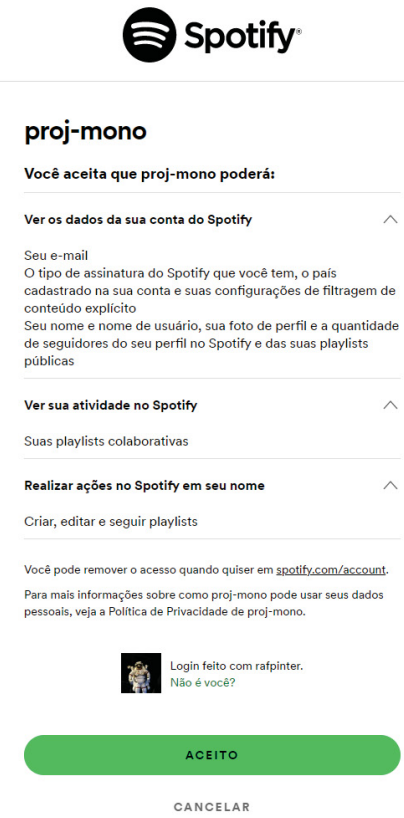
[15] G. Faggioli, M. Polato, F. Aiolli, *Efficient Similarity Based Methods For the Playlist Continuation Task*, (Association for Computing Machinery), (2018).

A. Telas da interface desenvolvida para a continuação de playlists

A.1. Página de *log in* na interface


RECOMMENDER SYSTEM
LOGIN WITH SPOTIFY

A.2. Página de *log in* com Spotify



A.3. Página principal da interface: aba de visualizar recomendações e exportar playlist

RECOMMENDER SYSTEM

Playlist URL: https://open.spotify.com/playlist/3Ai69qwZwUmAsNQTizpInO?si=1a28481de74c4a5c

Database percentage: 100%

Number of recommendations: 20

Factors: 100

Regularization: 1

Iterations: 10

Alpha: 1

TRAIN

Recommender | Playlists metrics | Explore | Distribution | Raw data

ROCKZIN

CREATED BY RAFPINTER
5 FOLLOWERS
60 TRACKS
AQUELE ROCKZIN CLÁSSICO

MATCHED SONGS

Song: Ramble On
Artist: Led Zeppelin
Album: Led Zeppelin II

Song: Good Times Bad Times
Artist: Led Zeppelin
Album: Led Zeppelin

Song: Travelling Riverside Blues (BBC Session)
Artist: Led Zeppelin
Album: Coda

Song: Sultans Of Swing
Artist: Dire Straits
Album: Dire Straits

Song: Walk Of Life
Artist: Dire Straits
Album: Brothers In Arms

Song: Burnin' for You
Artist: Blue Öyster Cult
Album: Fire of Unknown Origin

Song: This Charming Man - 2011 Remastered Version
Artist: The Smiths
Album: The Smiths

Song: Lovesong
Artist: The Cure
Album: Disintegration

Song: Bigmouth Strikes Again - 2011 Remastered Version
Artist: The Smiths

RECOMMENDED SONGS

Song: Take On Me
Artist: a-ha
Album: Hunting High And Low

Song: Don't You (Forget About Me)
Artist: Simple Minds
Album: Once Upon A Time

Song: Come On Eileen
Artist: Dexys Midnight Runners
Album: Too Rye Ay

Song: Everybody Wants To Rule The World
Artist: Tears For Fears
Album: Songs From The Big Chair

Song: Your Love
Artist: The Outfield
Album: Play Deep

Song: You Make My Dreams - Remastered
Artist: Daryl Hall & John Oates
Album: Voices

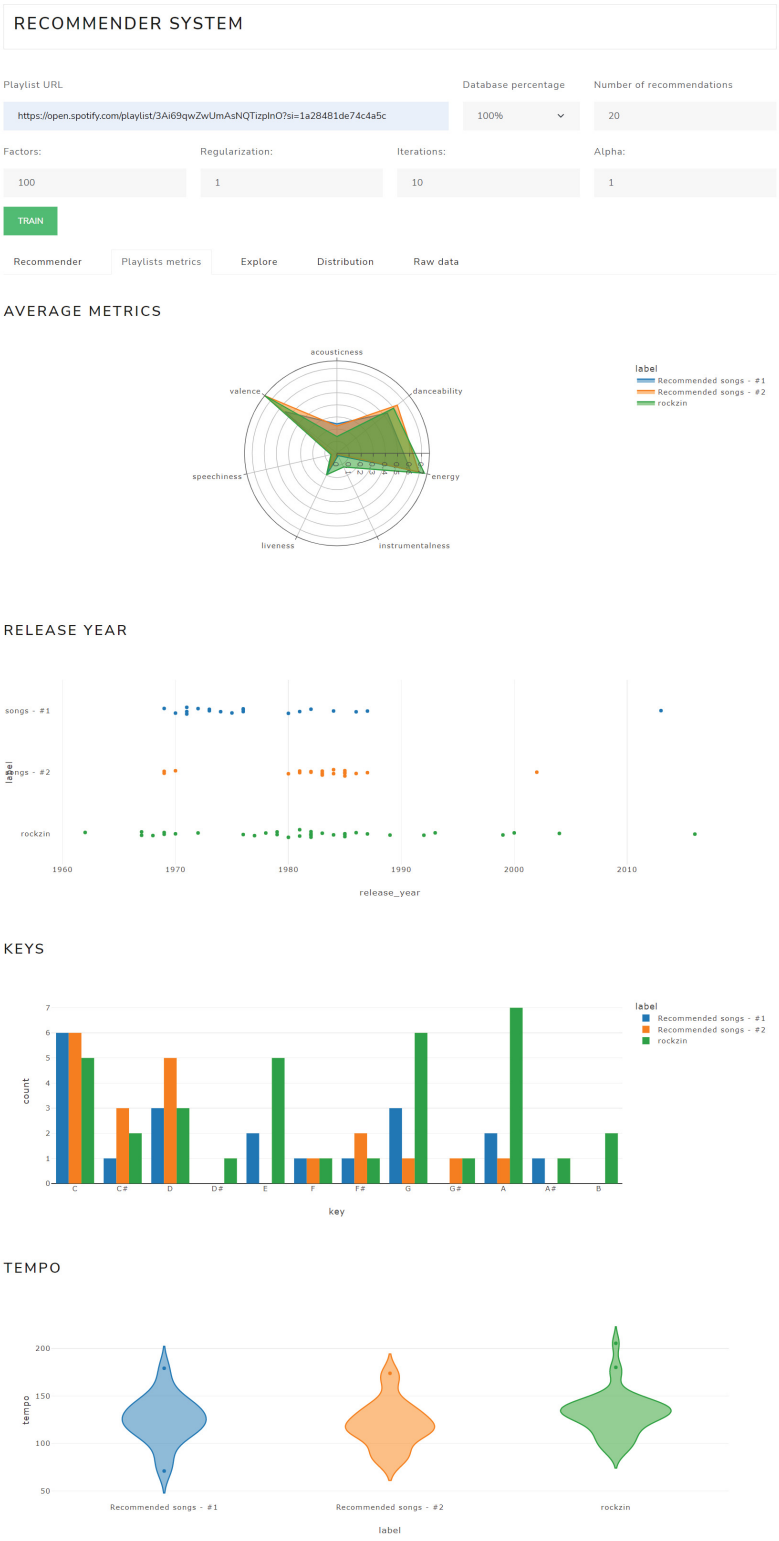
Song: Tainted Love
Artist: Soft Cell
Album: Non-Stop Erotic Cabaret

Song: Africa
Artist: Toto
Album: Toto IV

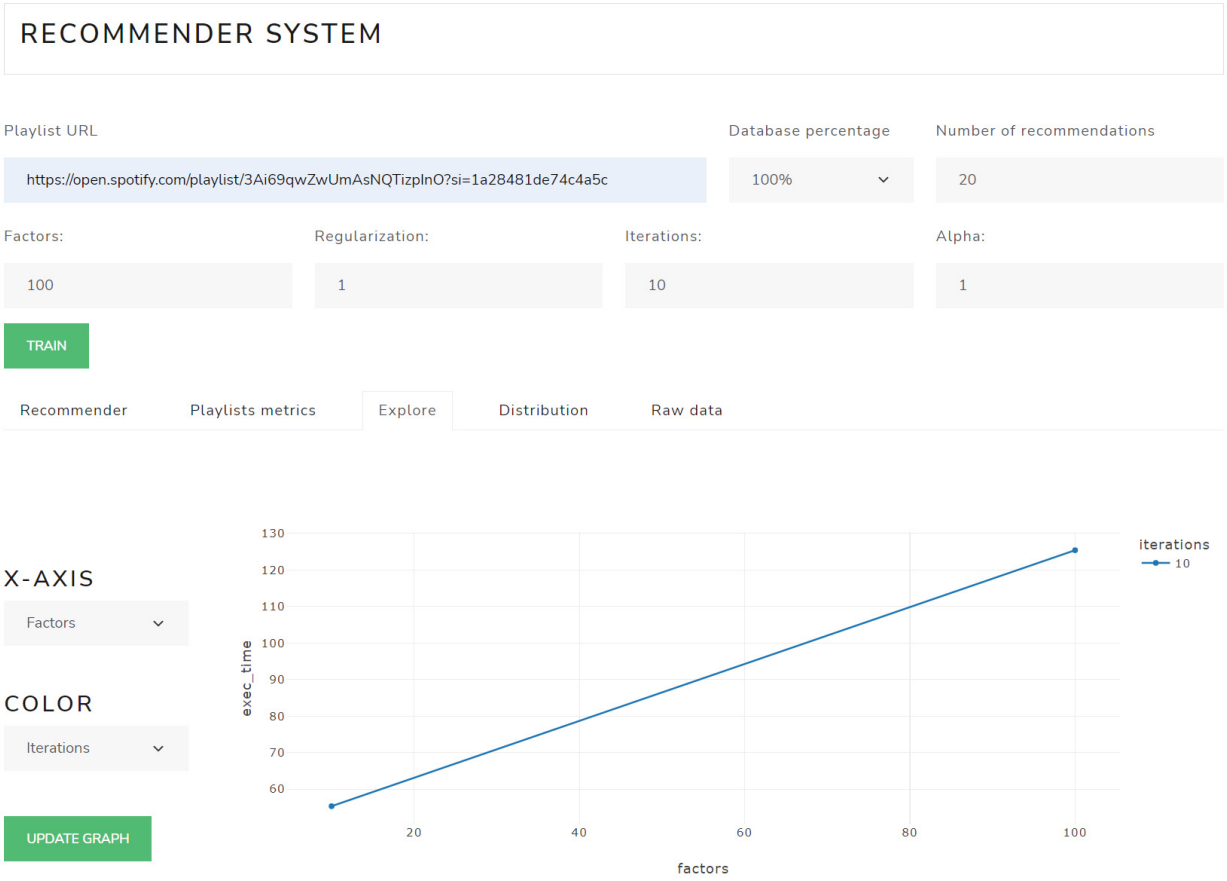
Song: Fortunate Son
Artist: Creedence Clearwater Revival

EXPORT PLAYLIST TO SPOTIFY

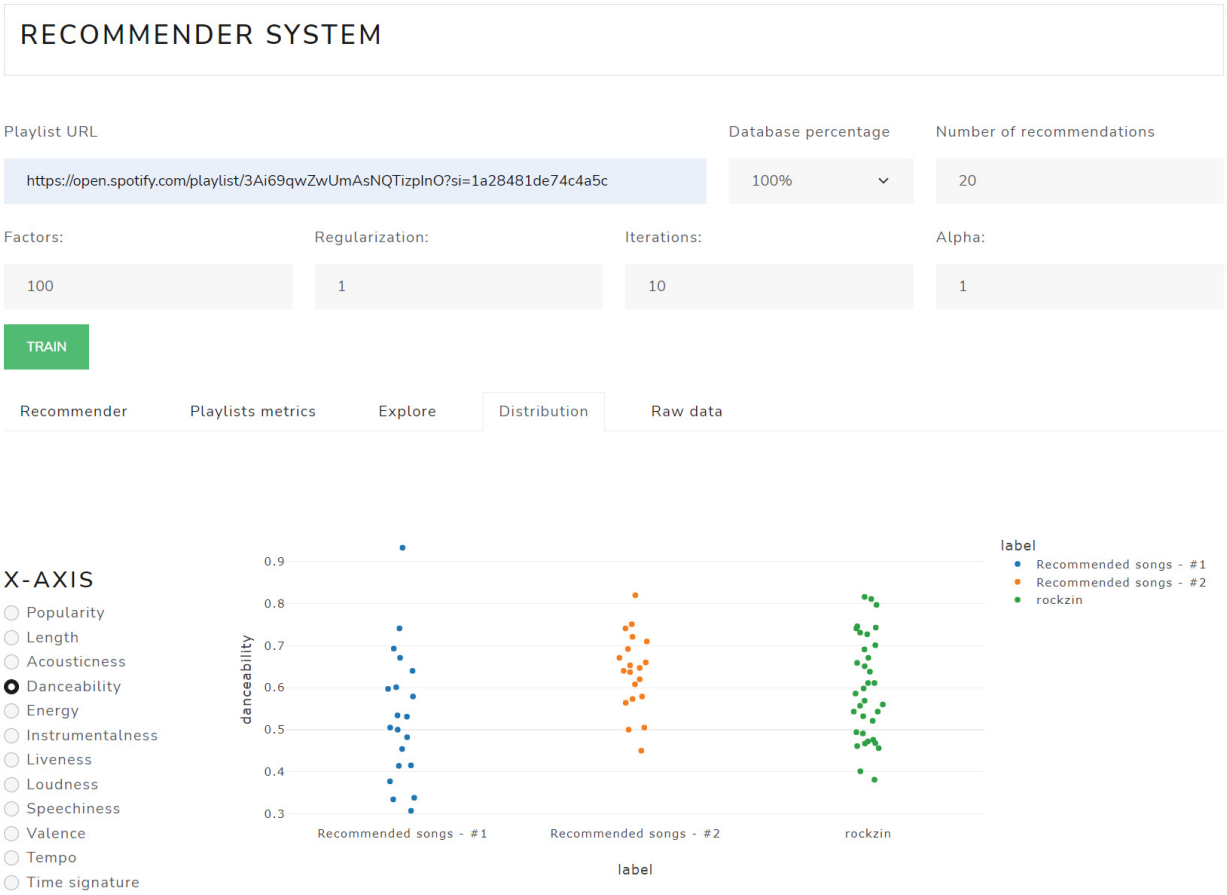
A.4. Página principal da interface: aba de comparar métricas gerais da playlist e recomendação



A.5. Página principal da interface: aba de explorar tempos de execução nos cálculos em função dos hiperparâmetros escolhidos



A.6. Página principal da interface: aba de visualizar distribuição das métricas da playlist e recomendações



A.7. Página principal da interface: aba de visualizar dados crus e exportar conjunto de dados

RECOMMENDER SYSTEM

Playlist URL: https://open.spotify.com/playlist/3Ai69qwZwUmAsNQTizplnO?si=1a28481de74c4a5c

Database percentage: 100%

Number of recommendations: 20

Factors: 100

Regularization: 1

Iterations: 10

Alpha: 1

TRAIN

Recommender | Playlists metrics | Explore | Distribution | Raw data

TRAINING PARAMETERS

LABEL	FACTORS	ITERATIONS	REGULARIZATION	DB_FRACTION	ALPHA	EXEC_TIME
Recommended songs - #1	10	10	1	1	1	55.33731985092163
Recommended songs - #2	100	10	1	1	1	125.39154958724976

RAW DATA

LABEL	ARTIST	NAME	ALBUM
Recommended songs - #2	a-ha	Take on Me	Hunting High and Low
Recommended songs - #2	Simple Minds	Don't You (Forget About Me)	Once Upon A Time (Deluxe)
Recommended songs - #2	Dexys Midnight Runners	Come On Eileen	Too Rye Ay
Recommended songs - #2	Tears For Fears	Everybody Wants To Rule The World	Songs From The Big Chair
Recommended songs - #2	The Outfield	Your Love	Play Deep
Recommended songs - #2	Daryl Hall & John Oates	You Make My Dreams (Come True)	Voices
Recommended songs - #2	Soft Cell	Tainted Love	Non-Stop Erotic Cabaret
Recommended songs - #2	TOTO	Africa	Toto IV
Recommended songs - #2	Creedence Clearwater Revival	Fortunate Son	Willy And The Poor Boys (Expanded Edition)
Recommended songs - #2	Wham!	Wake Me Up Before You Go-Go	Make It Big
Recommended songs - #2	Bryan Adams	Summer Of '69	Reckless
Recommended songs - #2	Eurythmics	Sweet Dreams (Are Made of This) - Remastered	Sweet Dreams (Are Made Of This)
Recommended songs - #2	Creedence Clearwater Revival	Bad Moon Rising	Green River (40th Anniversary Edition)
Recommended songs - #2	Kenny Loggins	Footloose - From "Footloose" Soundtrack	The Essential Kenny Loggins
Recommended songs - #2	Eddie Money	Take Me Home Tonight	Can't Hold Back
Recommended songs - #2	Cyndi Lauper	Girls Just Want to Have Fun	She's So Unusual
Recommended songs - #2	The Police	Every Breath You Take - Remastered 2003	Synchronicity (Remastered)
Recommended songs - #2	Creedence Clearwater Revival	Have You Ever Seen The Rain	Pendulum (40th Anniversary Edition)
Recommended songs - #2	The J. Geils Band	Centerfold	Freeze Frame
Recommended songs - #2	Rick Astley	Never Gonna Give You Up	Whenever You Need Somebody

« ‹ 2 / 2 › »

DOWNLOAD RAW DATA

DOWNLOAD EXECUTION TIME DATA