

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

SAN CUNHA DA SILVA
Orientador: Prof. Dr. Reinaldo Silva Fortes

**APRIMORANDO A INSTALAÇÃO E A CONFIGURAÇÃO DE
EXPERIMENTOS DO RECSYSEXP**

Ouro Preto, MG
2024

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

SAN CUNHA DA SILVA

**APRIMORANDO A INSTALAÇÃO E A CONFIGURAÇÃO DE EXPERIMENTOS DO
RECSYSEXP**

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Reinaldo Silva Fortes

Ouro Preto, MG
2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S586a Silva, San Cunha da.
Aprimorando a instalação e a configuração de experimentos do
Recsysexp. [manuscrito] / San Cunha da Silva. - 2024.
33 f.: il.: color., gráf., tab.. + Algoritmos.

Orientador: Prof. Dr. Reinaldo Silva Fortes.
Monografia (Bacharelado). Universidade Federal de Ouro Preto.
Instituto de Ciências Exatas e Biológicas. Graduação em Ciência da
Computação .

1. Ciência da computação. 2. Sistemas de recomendação. 3.
Foundation (Framework). 4. Interface gráfica com o usuário (Sistemas de
computação). I. Fortes, Reinaldo Silva. II. Universidade Federal de Ouro
Preto. III. Título.

CDU 681.32

Bibliotecário(a) Responsável: Paulo Vitor Oliveira - CRB6/2551



FOLHA DE APROVAÇÃO

San Cunha da Silva

Aprimorando a instalação e a configuração de experimentos do RecSysExp

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 22 de Fevereiro de 2024.

Membros da banca:

Reinaldo Silva Fortes (Orientador) - Doutor - Universidade Federal de Ouro Preto
Anderson Almeida Ferreira (Examinador) - Doutor - Universidade Federal de Ouro Preto
Pedro Henrique Lopes Silva (Examinador) - Doutor - Universidade Federal de Ouro Preto

Reinaldo Silva Fortes, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 22/02/2024.



Documento assinado eletronicamente por **Reinaldo Silva Fortes, PROFESSOR DE MAGISTERIO SUPERIOR**, em 22/02/2024, às 15:09, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0667225** e o código CRC **C47C0F5D**.

Dedico este trabalho à minha família, Uédson, Consuelo e Sandy, por nunca desistirem de mim e pelo suporte até o final.

Agradecimentos

Primeiramente, agradeço aos meus pais Uédson e Consuelo por nunca medirem esforços para que fosse possível realizar esse sonho. Em seguida, expresso minha gratidão à minha irmã Sandy por todo o companheirismo e ideias trocadas ao longo da faculdade e da vida. Também quero agradecer a Davidson, John, Tairan e Vitim por tornarem as voltas para casa memoráveis e as despedidas mais leves. Um sincero agradecimento a todos os amigos que fiz nessa jornada e que de alguma forma contribuíram para a finalização deste trabalho. Agradeço ao professor Reinaldo pela oportunidade, por toda compreensão e pelos ensinamentos recebidos. Por último, mas não menos importante, agradeço à República Sem Destino pelos anos de acolhimento e por sempre fornecer o melhor suporte possível.

A tecnologia de reutilização ideal fornece componentes que podem ser facilmente conectados para criar um novo sistema ([JOHNSON, 1997](#)).

Resumo

O trabalho apresenta aprimoramentos significativos no *framework* RecSysExp, utilizado para execução de experimentos em sistemas de recomendação. Estas melhorias foram direcionadas para aprimorar a usabilidade, escalabilidade e legibilidade do sistema. As novas funcionalidades abrangem três áreas distintas: o desenvolvimento de uma interface gráfica, o encapsulamento do *framework* utilizando *Docker*, e a reestruturação de uma classe para integração mais coesa com conjuntos de dados, seguindo padrões de projeto estabelecidos. O objetivo primordial foi elevar o valor proporcionado pelo *framework*, alinhado com a visão dos seus criadores, visando seu uso como ferramenta acadêmica em ambientes de sala de aula ou pesquisa. A abordagem metodológica adotada utilizou tecnologias específicas para cada contexto abordado. Para a criação da interface do usuário, foram empregados *React* e *Next.js*, *frameworks* de *frontend*, enquanto para o encapsulamento do RecSysExp, foram utilizados *Dockerfile* e *docker-compose*. Por fim, a modificação da classe responsável pelos *datasets* foi realizada seguindo o padrão de projeto *Template Method*. O projeto atingiu com sucesso todos os objetivos propostos. A implementação de uma estrutura de contêineres simplificou a instalação do sistema, enquanto melhorias na visualização das configurações tornaram a criação de experimentos mais intuitiva. Além disso, a capacidade de enviar arquivos expandiu as opções dos usuários. Embora a versão final do RecSysExp funcione de forma semelhante à sua iteração original, as adições deste trabalho resultaram em uma versão aprimorada e mais amigável. No entanto, é importante destacar que a configuração através da interface gráfica possui limitações, pois só é possível configurar algoritmos e módulos que podem ser instanciados via arquivo de configuração no *framework*. Algoritmos e módulos implementados apenas como bibliotecas em outros projetos não podem ser configurados via *frontend*.

Palavras-chave: Sistemas de Recomendação. Frameworks. RecSysExp.

Abstract

The paper presents significant enhancements to the RecSysExp framework, used for conducting experiments in recommendation systems. These improvements were aimed at enhancing the usability, scalability, and readability of the system. The new functionalities cover three distinct areas: the development of a graphical user interface, the encapsulation of the framework using Docker, and the restructuring of a class for more cohesive integration with datasets, following established design patterns. The primary goal was to enhance the value provided by the framework, aligned with the vision of its creators, aiming at its use as an academic tool in classroom or research environments. The methodological approach adopted employed specific technologies for each addressed context. For the creation of the user interface, React and Next.js frontend frameworks were employed, while Dockerfile and docker-compose were used for the encapsulation of RecSysExp. Finally, the modification of the class responsible for datasets was carried out following the Template Method design pattern. The project successfully achieved all proposed objectives. The implementation of a container structure simplified the installation of the system, while improvements in the visualization of configurations made experiment creation more intuitive. Additionally, the ability to upload files expanded user options. Although the final version of RecSysExp functions similarly to its original iteration, the additions from this work resulted in an enhanced and more user-friendly version. However, it is important to note that configuration through the graphical interface has limitations, as it is only possible to configure algorithms and modules that can be instantiated via configuration files in the framework. Algorithms and modules implemented solely as libraries in other projects cannot be configured via the frontend.

Keywords: Recommender Systems. Frameworks. RecSysExp.

Lista de Ilustrações

| | |
|---|----|
| Figura 3.1 – Tela inicial. | 16 |
| Figura 3.2 – Campos de Configuração de <i>Dataset</i> | 16 |
| Figura 3.3 – Campos de Configuração de <i>Preprocessing</i> | 16 |
| Figura 3.4 – Campos de Configuração de <i>Metrics</i> | 17 |
| Figura 3.5 – Campos de Configuração de <i>Recommenders</i> | 17 |
| Figura 3.6 – Campos de Configuração de <i>Visualization</i> | 17 |
| Figura 3.7 – Visualização do experimento montado | 18 |
| Figura 4.1 – Resposta do Servidor Sobre <i>Status</i> | 25 |
| Figura 4.2 – <i>Upload MovieLens</i> | 26 |
| Figura 4.3 – Configuração de <i>Preprocessing</i> | 26 |
| Figura 4.4 – Configuração de <i>Metrics</i> | 27 |
| Figura 4.5 – Configuração de <i>Recommenders</i> | 27 |
| Figura 4.6 – Configuração de <i>Visualization</i> | 27 |
| Figura 4.7 – <i>Ratings</i> por Usuário | 29 |
| Figura 4.8 – Itens mais avaliados | 29 |

Lista de Tabelas

| | |
|--|----|
| Tabela 4.1 – Comparação de resultados obtidos. | 28 |
|--|----|

Lista de Algoritmos

| | | |
|-----|---|----|
| 3.1 | <i>Dockerfile</i> RecSysExp | 19 |
| 3.2 | <i>Dockerfile</i> UI | 20 |
| 3.3 | <i>docker-compose</i> | 22 |
| 3.4 | Objeto de configuração do <i>framework</i> original | 23 |
| 3.5 | Classe <i>GeneralDataset</i> | 23 |

Lista de Abreviaturas e Siglas

| | |
|-------|--|
| ABNT | Associação Brasileira de Normas Técnicas |
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| AUC | Area Under the ROC Curve |
| CBF | Content-Based Filtering |
| CF | Collaborative Filtering |
| DECOM | Departamento de Computação |
| DOM | Document Object Model |
| HF | Hybrid Filtering |
| HTML | Hypertext Markup Language |
| IoT | Internet of Things |
| MAE | Mean Absolute Error |
| MAP | Mean Average Precision |
| MAPE | Mean Percentage Error |
| MRR | Mean Reciprocal Rank |
| MTJ | Matrix Toolkit for Java |
| NDCG | Normalized Discounted Cumulative Gain |
| RMSE | Root Mean Square Error |
| RS | Recommendation Systems |
| SPA | Single Page Application |
| UI | User Interface |
| UFOP | Universidade Federal de Ouro Preto |
| UX | User Experience |

Sumário

| | | |
|----------|--|-----------|
| 1 | Introdução | 1 |
| 1.1 | Justificativa | 2 |
| 1.2 | Objetivos | 3 |
| 1.3 | Organização do Trabalho | 3 |
| 2 | Revisão Bibliográfica | 4 |
| 2.1 | Sistemas de Recomendação | 4 |
| 2.1.1 | Filtragem Colaborativa | 5 |
| 2.1.2 | Filtragem Baseado em Conteúdo | 5 |
| 2.1.3 | Filtragem Híbrida | 6 |
| 2.2 | <i>Docker</i> | 7 |
| 2.3 | Single Page Application | 7 |
| 2.3.1 | <i>React</i> | 8 |
| 2.3.2 | <i>Next.js</i> | 8 |
| 2.4 | Padrões de Projetos | 8 |
| 2.5 | Trabalhos Relacionados | 9 |
| 2.5.1 | RecSysExp | 10 |
| 2.5.2 | RecMetrics | 10 |
| 2.5.3 | LibRec | 11 |
| 2.5.4 | Lenskit | 12 |
| 2.5.5 | Xperimentor | 13 |
| 2.5.6 | Discussão | 13 |
| 3 | Desenvolvimento | 15 |
| 3.1 | Criação da Interface Gráfica | 15 |
| 3.2 | Encapsulamento usando <i>Docker</i> | 18 |
| 3.3 | Alteração na Arquitetura da Classe <i>Datasets</i> | 22 |
| 4 | Resultados | 25 |
| 4.1 | Criação das imagens | 25 |
| 4.2 | Inicialização do sistema | 25 |
| 4.3 | <i>Upload</i> do arquivo | 26 |
| 4.4 | Configuração do Experimento | 26 |
| 4.5 | Comparação dos Resultados | 26 |
| 5 | Considerações Finais | 30 |
| 5.1 | Conclusão | 30 |
| 5.2 | Trabalhos Futuros | 30 |

Referências 32

1 Introdução

A tarefa dos sistemas de recomendação (RS, do inglês *Recommender Systems*) é transformar dados sobre os usuários e suas preferências em previsões de possíveis gostos e interesses futuros dos usuários. O estudo dos sistemas de recomendação está numa encruzilhada da ciência e da vida socioeconômica, o seu enorme potencial foi notado pela primeira vez pelos empreendedores da web na vanguarda da revolução da informação (LÜ et al., 2012). Atualmente com grande parte de nossos dispositivos conectados entre si, geramos dados de várias formas, dados esses que alimentam sistemas de recomendação dos mais variados modelos. É através desse processo que nos são apresentadas novas músicas, novos filmes, novas notícias e uma infinidade de possibilidades.

O termo *framework* já foi utilizado em vários contextos da computação ao longo de toda a evolução da tecnologia. Atualmente podemos definir um *framework* como uma estrutura pré-projetada e reutilizável que fornece uma base para o desenvolvimento de aplicativos de software. Por ter uma definição tão abrangente, consequentemente tem uma área de atuação tão grande quanto. Existem *frameworks* para diversas linguagens e diversos propósitos, todos de certa forma seguem um caminho para facilitar o desenvolvimento de softwares. Os *frameworks* normalmente encapsulam funcionalidades comuns, padrões de projetos e melhores práticas, permitindo que os desenvolvedores se concentrem na implementação das características únicas de seu projeto.

Este trabalho parte da monografia de Natali (2023), que apresenta um *framework* de alto nível de abstração para implementação e validação de sistemas de recomendação. Visando permitir a execução de experimentos utilizando de diferentes base de dados, pré-processamentos, recomendadores, formas de avaliação e também diferentes visualizações dos dados, sejam eles de entrada ou de saída, o **RecSysExp** foi criado. Seguindo a definição, o *framework* criado encapsula outros projetos dentro de si para que o desenvolvedor tenha todas as ferramentas no mesmo lugar e uma facilidade para testar novas formas de criar, executar e replicar experimentos de sistemas de recomendação. Após todo o ambiente configurado, o *framework* só precisa de uma linha de comando para execução de diversos experimentos, todos orquestrados de maneira independente e transparente para o usuário.

Tendo o **RecSysExp** em mãos, esse trabalho propõe três ajustes para que o *framework* se torne mais acessível e ganhe usabilidade na hora da execução. A primeira proposta é a criação de um interface gráfica para que o usuário tenha uma maior facilidade de configurar seus experimentos e, consequentemente, uma melhor experiência com a ferramenta. A segunda proposta é containerizar o *framework* utilizando *Docker* para que não haja problemas nas configurações de dependências necessárias. Por fim, realizar alterações na interface de *datasets* para que seja possível utilizar uma gama maior de bases de dados.

A metodologia adotada centrou-se no desenvolvimento de novas funcionalidades destinadas a serem incorporadas ao *framework*, alinhadas aos objetivos estabelecidos. Para abordar a criação da Interface do Usuário (UI, do inglês *User Interface*), foi implementada uma página web utilizando *React* e *Next.js*, proporcionando a visualização de todas as opções de configuração aceitas pelo **RecSysExp**. Embora o foco principal seja no encapsulamento do *framework*, a parte gráfica também foi containerizada. Para isso, foi criado um *Dockerfile* para cada aplicação, garantindo a satisfação dos requisitos do sistema e das dependências específicas de cada projeto. Com as imagens prontas, foi elaborado um *docker-compose* para consolidar e iniciar conjuntamente o projeto como um todo. As alterações no código fonte original, foram a reformulação da estrutura de classe responsável por receber as bases de dados e adição de uma camada *web* para receber requisições. Foi realizada uma refatoração do código, adotando o padrão de projeto conhecido como *Template Method*, aprimorando a modularidade e a manutenibilidade do sistema.

O trabalho alcançou com sucesso todos os objetivos. A implementação de uma estrutura de contêineres que inicializa todos os componentes do sistema simultaneamente simplifica significativamente a instalação para os usuários. Além disso, a melhoria da visualização das configurações disponíveis torna a criação de experimentos mais intuitiva e assertiva, enquanto a capacidade de enviar arquivos expande as opções do usuário. Embora a versão final do **RecSysExp** funcione de forma semelhante à sua iteração original, as adições deste trabalho resultaram em uma versão aprimorada e mais amigável. A conclusão reforça que as melhorias propostas foram benéficas, destacando, no entanto, que a versão original do *framework* possui limitações na configuração através do arquivo inicial devido à natureza dos módulos e algoritmos implementados, que são utilizáveis apenas como bibliotecas em outros projetos, não permitindo configuração e execução por meio de um arquivo de configuração, essas limitações são refletidas neste trabalho.

Nas seções seguintes, são apresentados as justificativas (Seção 1.1) e os objetivos (Seção 1.2) deste trabalho. As justificativas são apresentadas para contextualizar a relevância e a necessidade do projeto. Em seguida, os objetivos são para fornecer uma visão clara das metas específicas que o estudo pretende alcançar.

1.1 Justificativa

Esse trabalho se justifica por ser a implementação de pontos importantes sugeridos como trabalhos futuros nas conclusões de Natali (2023). Buscando agregar ao campo dos sistemas de recomendação, as melhorias propostas por esse trabalho tornam a utilização e a compreensão do **RecSysExp** muito mais fácil, podendo assim atrair novas pessoas interessadas a desenvolver ou conhecer o ecossistema.

A validação das melhorias propostas neste trabalho justifica sua implementação. Um resultado positivo não apenas valida as abordagens propostas, mas também abre caminho para futuros desenvolvimentos, utilizando os *insights* e resultados obtidos como base para evoluções.

Assim como um resultado negativo levaria a uma nova abordagem na forma de expandir o *framework*.

1.2 Objetivos

O objetivo geral desse trabalho é estender os estudos e esforços de [Natali \(2023\)](#) de uma forma que o material já produzido seja aproveitado da melhor maneira possível e coloque o **RecSysExp** disponível para qualquer nível de usuário se aventurar no campo dos sistemas de recomendação de forma mais simples e direta.

Para alcançar o objetivo principal, há três passos que podem ser considerados os pilares da execução desse projeto:

1. Criar de uma interface gráfica capaz de prover para o usuário todas as ferramentas à sua disposição, tornando a configuração muito mais simples e intuitiva, melhorando consequentemente a experiência do usuário;
2. Encapsular todas as dependências de execução (inclusive a interface gráfica) dentro de *containers* para que não haja problemas relacionados ao sistema operacional do usuário e tornando a instalação simples e rápida;
3. Aplicar padrões de projetos na classe referente aos *datasets* para que seja possível executar experimentos utilizando arquivos enviados pelo usuário.

1.3 Organização do Trabalho

Até aqui, no Capítulo 1, foram descritos brevemente todos os pontos que serão abordados nesse trabalho, a justificativa e os objetivos da realização do mesmo. A partir daqui, as informações seguem a ordem descrita a seguir.

O Capítulo 2 é dividido em duas partes, sendo a primeira entre as Seções 2.1 e 2.4 que contém detalhes sobre os principais tópicos aqui mencionados, como sistemas de recomendação, *docker*, padrões de projetos entre outros, e a segunda parte, na Seção 2.5 referente aos trabalhos relacionados.

Seguindo para o Capítulo 3 há o detalhamento técnico de como os três objetivos propostos foram implementados. Logo em seguida, no Capítulo 4, são apresentados os resultados encontrados após a finalização do trabalho. E por fim, no Capítulo 5, são apresentadas a conclusão e ideias para trabalhos futuros.

2 Revisão Bibliográfica

Neste capítulo, serão abordados todos os conceitos e tecnologias empregados no desenvolvimento do projeto. Na Seção 2.1, encontramos a definição fundamental dos sistemas de recomendação, abrangendo as Subseções 2.1.1, 2.1.2 e 2.1.3, que tratam, respectivamente, da Filtragem Colaborativa (CF, do inglês *Collaborative Filtering*), Filtragem Baseada em Conteúdo (CBF, do inglês *Content-Based Filtering*) e Filtragem Híbrida (HF, do inglês *Hybrid Filtering*). As seções subsequentes exploram as ideias e o funcionamento das ferramentas utilizadas para implementar as novas funcionalidades do sistema, sendo elas: *Docker* na Seção 2.2, *Single Page Application* na Seção 2.3 e padrões de projetos na Seção 2.4.

Na Seção 2.5, é apresentada uma série de trabalhos relacionados, trazendo ideias e discussões que merecem aprofundamento. Este panorama ampliado contribuirá para situar o presente trabalho no contexto das pesquisas e desenvolvimentos já realizados na área.

2.1 Sistemas de Recomendação

Sistemas de Recomendação têm sido aplicados em diversos setores, como comércio eletrônico, saúde, redes sociais, indústria, e-learning, música, internet das coisas (IoT, do inglês *Internet of Things*), sistema de informação alimentar e nutricional, e marketing. Eles aprendem com as experiências e opiniões dos usuários, utilizando o comportamento destes para recomendar itens ou produtos considerados mais relevantes dentre as opções disponíveis. Além disso, esses sistemas proporcionam facilidades para aprimorar a adaptação das aplicações a cada usuário. No contexto do comércio eletrônico, os sistemas de recomendação automatizam a personalização do ambiente, incorporando tanto técnicas tradicionais quanto modernas, como as técnicas de aprendizado de máquina (ALAMDARI et al., 2020). Os RS desempenham um papel crucial na era da informação, auxiliando os usuários a descobrirem itens relevantes em meio a uma vasta quantidade de opções.

Sistemas de Recomendação utilizam diversas fontes de informação para fornecer aos usuários previsões e recomendações de itens. Eles buscam equilibrar fatores como precisão, novidade, dispersão e estabilidade nas recomendações (BOBADILLA et al., 2013). Os sistemas de recomendação são mecanismos projetados para prever as preferências do usuário e fornecer sugestões personalizadas. Inicialmente, eram focados em Filtragem Colaborativa (Subseção 2.1.1) e Baseada em Conteúdo (Subseção 2.1.2). No entanto, ao longo do tempo, surgiram outras abordagens, como as Híbridas (Subseção 2.1.3) que combinam elementos de diferentes técnicas.

2.1.1 Filtragem Colaborativa

A Filtragem Colaborativa analisa o comportamento passado e as preferências de usuários semelhantes para fazer recomendações. Pode ser dividida em filtragem colaborativa usuário-usuário e item-item, cada uma com suas vantagens e desafios.

Ao discutir CF, destacam-se alguns pontos positivos. Por exemplo, não é necessário que o modelo possua conhecimento prévio do domínio da aplicação, pois as incorporações são aprendidas automaticamente. Essa característica é particularmente vantajosa, pois elimina a necessidade de uma compreensão profunda do contexto específico. Além disso, uma vantagem significativa da CF é sua capacidade de recomendar itens mesmo quando o modelo não tem certeza se um usuário está interessado neles. Isso é possível ao basear as recomendações no interesse de usuários semelhantes (SCHAFER et al., 2007).

No entanto, a Filtragem Colaborativa enfrenta desafios específicos, como a dificuldade em lidar com novos itens. Esse desafio é conhecido como o problema de inicialização a frio, que se refere à limitação do sistema em fazer inferências para itens ou usuários sobre os quais não possui informações suficientes. Isso destaca uma das limitações da CF, especialmente quando se trata de integrar novos elementos ao sistema (DEVELOPERS, 2022).

Adicionalmente, vale ressaltar que a Filtragem Colaborativa é amplamente empregada em diversas áreas, incluindo comércio eletrônico, *streaming* de conteúdo, e serviços online, demonstrando sua relevância e aplicabilidade em cenários diversos (NATALI, 2023).

As duas áreas principais da Filtragem Colaborativa são os métodos de vizinhança e os modelos de fatores latentes. Os métodos de vizinhança têm como foco calcular as relações entre itens ou, alternativamente, entre usuários. A abordagem orientada para itens avalia a preferência de um usuário por um item com base nas classificações de itens “vizinhos” pelo mesmo usuário. Os vizinhos de um item são outros itens que costumam receber classificações semelhantes quando avaliados pelo mesmo usuário. Por exemplo, considere o filme “Resgate do Soldado Ryan”. Seus vizinhos podem incluir filmes de guerra, filmes de Spielberg e filmes de Tom Hanks, entre outros. Para prever a avaliação de um usuário específico para “Resgate do Soldado Ryan”, procuraríamos pelos vizinhos mais próximos do filme que esse usuário realmente avaliou (KOREN; BELL; VOLINSKY, 2009).

2.1.2 Filtragem Baseado em Conteúdo

Os sistemas de recomendação baseados em conteúdo são utilizados na recuperação de informações. Inicialmente, termos são atribuídos manualmente, o que significa que, durante a atribuição manual de termos, uma técnica deve ser escolhida para comparar esses termos com as informações no perfil do cliente. Além disso, um algoritmo de aprendizado deve ser escolhido para executar essas técnicas e, em seguida, trazer os resultados relacionados ao cliente.

Os conceitos de frequência de termos e frequência inversa de documento são utilizados em

sistemas de recuperação de informações e em sistemas de filtragem baseados em conteúdo (como os recomendadores baseados em conteúdo). Eles são empregados para determinar a importância relativa de um documento, entre outros. A filtragem baseada em conteúdo é um método popular no design de sistemas de recomendação. Os métodos de filtragem baseada em conteúdo são fundamentados em uma descrição do objeto e em um perfil das preferências do usuário. Em um sistema de recomendação baseado em conteúdo, palavras-chave são usadas para descrever os itens, e um perfil pessoal é criado para sugerir o tipo de item que esse usuário gosta. Em outras palavras, esses algoritmos tentam recomendar objetos semelhantes aos que o usuário apreciou no passado (ou está examinando no presente). Vários itens candidatos são comparados com itens previamente avaliados pelo usuário, e os itens com os melhores resultados de correspondência são recomendados (JAVED et al., 2021).

Este tipo de sistema recomenda itens com base nas características e descrições dos itens e nas preferências históricas do usuário. A análise de conteúdo é fundamental para identificar padrões relevantes.

Ao explorar a CBF, destacam-se alguns benefícios. Por exemplo, essa abordagem não requer informações sobre o comportamento ou preferências de outros usuários, o que pode ser vantajoso em situações em que dados de colaboração são limitados ou indisponíveis. Além disso, a CBF é menos suscetível ao problema de inicialização a frio, pois pode recomendar itens com base nas características intrínsecas dos itens, mesmo que haja pouca informação sobre a preferência do usuário.

Entretanto, a CBF também enfrenta desafios específicos, como a necessidade de um bom entendimento e extração das características relevantes dos itens. Isso pode ser crucial para garantir recomendações precisas e relevantes. Além disso, a CBF pode ter dificuldades em lidar com a evolução das preferências do usuário ao longo do tempo, já que se baseia principalmente em características estáticas dos itens. (METEREN; SOMEREN, 2000)

2.1.3 Filtragem Híbrida

Sistemas de recomendação híbridos combinam duas ou mais técnicas de recomendação para obter um desempenho melhor com menos das desvantagens de cada uma individualmente. Mais comumente, a filtragem colaborativa é combinada com alguma outra técnica na tentativa de evitar o problema de inicialização a frio (BURKE, 2002).

Uma das vantagens da Filtragem Híbrida é a integração de múltiplos métodos de recomendação permitindo que o sistema obtenha benefícios das vantagens individuais de cada abordagem, ao mesmo tempo em que mitiga suas limitações. Essa sinergia possibilita um desempenho mais robusto, especialmente em cenários complexos ou com dados escassos. Além disso, a HF é capaz de lidar de forma mais eficiente com problemas como o inicialização a frio e a escassez de dados de colaboração.

No entanto, como desvantagens temos a necessidade de gerenciar a complexidade resultante da combinação de diferentes modelos. A seleção e a ponderação adequadas dos métodos de filtragem colaborativa e baseada em conteúdo são essenciais para otimizar o desempenho global do sistema híbrido. Além disso, a implementação eficaz da HF demanda um entendimento profundo das características dos itens, das preferências dos usuários e das dinâmicas do sistema.

Existem várias abordagens para a hibridização de sistemas de recomendação, categorizadas em diferentes estratégias. A Fusão Ponderada consiste em combinar as pontuações previstas pelos métodos de filtragem colaborativa e baseada em conteúdo por meio de uma ponderação, atribuindo pesos específicos a cada método. A Fusão de Resultados une as recomendações geradas separadamente por ambos os métodos, formando uma lista integrada. A Fusão de Nível de Característica realiza a hibridização em características específicas dos itens ou usuários, selecionando e combinando características seletivamente. A abordagem em cascata envolve a geração inicial de recomendações por um método, refinadas por outro método subsequente, como usar recomendações colaborativas como entrada para a filtragem baseada em conteúdo, proporcionando uma sequência de refinamento para melhorar a personalização das recomendações. (ISINKAYE; FOLAJIMI; OJOKOH, 2015)

2.2 Docker

Docker é uma plataforma de software que permite automatizar o processo de empacotamento, distribuição e execução de aplicativos em ambientes isolados chamados contêineres. O *Docker* utiliza a virtualização a nível de sistema operacional para criar contêineres, que são ambientes leves e isolados. Cada contêiner contém todos os recursos necessários para executar um aplicativo, incluindo código, bibliotecas e dependências.

Uma imagem *Docker* é um pacote independente que contém tudo o que é necessário para executar um aplicativo, incluindo o código, o ambiente de execução, as bibliotecas, as variáveis de ambiente e configurações. Um contêiner é uma instância de uma imagem. As imagens *Docker* são construídas em camadas. Cada instrução no *Dockerfile* (que é o nome do arquivo de configuração das imagens) cria uma camada, e o *Docker* utiliza um sistema de cache para otimizar a construção de imagens, reutilizando camadas existentes sempre que possível (MERKEL, 2014).

2.3 Single Page Application

Uma *Single Page Application* (SPA) é um tipo de aplicação web que interage com o usuário, fornecendo uma experiência contínua, sem a necessidade de recarregar a página inteira durante a navegação. Isso é possível pelo fato de uma SPA carregar uma única página HTML (*Hypertext Markup Language*) inicialmente e, em seguida, atualizar dinamicamente o conteúdo conforme o usuário interage, utilizando AJAX (*Asynchronous JavaScript and XML*) e manipulação do DOM

(*Document Object Model*). Em vez de depender de solicitações ao servidor para carregar novas páginas, as SPAs usam roteamento no lado do cliente para alterar dinamicamente as visualizações na própria página. Como benefícios temos uma resposta rápida, a experiência do usuário sem interrupções e a capacidade de criar interfaces mais complexas. (SMITH; JOHNSON, 2019)

Existem *frameworks* consolidados como *Angular*, *React* e *Vue.js* que oferecem ferramentas, funcionalidades e convenções que facilitam a construção de SPAs eficientes. Como exemplos de SPAs conhecidas temos *Gmail*, *Facebook*, *Outlook* e outros serviços de *e-mail*.

Nas subseções seguintes estão detalhados os dois *frameworks* usados na construção deste trabalho.

2.3.1 *React*

React é uma biblioteca *JavaScript* de código aberto utilizada para construir interfaces de usuário interativas e eficientes. Desenvolvida e mantida pelo Facebook, *React* se destaca por sua abordagem declarativa e por permitir a construção de componentes reutilizáveis. O estilo de desenvolvimento orientado a componentes do *React* incentiva a criação de interfaces divididas em componentes independentes.

Existem dois conceitos muito importantes desse *framework*. O estado é utilizado para armazenar dados que podem ser modificados ao longo do tempo, enquanto as propriedades (props) são utilizadas para passar dados de um componente pai para um componente filho (JACKSON, 2021).

2.3.2 *Next.js*

Next.js é um *framework React* de código aberto utilizado para construir aplicativos web modernos. Ele simplifica o processo de desenvolvimento, oferecendo recursos como roteamento automático, renderização do lado do servidor e do lado do cliente. É uma extensão natural do *React*, aproveitando todas as suas funcionalidades enquanto adiciona recursos para otimização e desenvolvimento eficiente.

O *Next.js* simplifica o roteamento de páginas em um aplicativo *React*, tornando-o automático e dinâmico. Cada componente *React* em um diretório *pages* se torna uma rota automaticamente, evitando assim todo o trabalho manual de declarar cada rota manualmente (VERCEL, 2021).

2.4 Padrões de Projetos

Os padrões de projeto têm suas raízes na obra de Christopher Alexander, um arquiteto que documentou uma série de padrões para construção de cidades e prédios em seu livro "A Patterns Language", de 1977. Esses padrões descrevem problemas recorrentes e suas soluções,

aplicáveis de maneira repetida. Posteriormente, em 1995, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides adaptaram essas ideias para o desenvolvimento de software em seu livro "Design Patterns: Elements of Reusable Object-Oriented Software". Eles definiram padrões de projeto como descrições de objetos e classes relacionados que resolvem problemas de projeto genéricos em um contexto particular. Esses padrões são compreendidos através do entendimento do problema a ser resolvido, o contexto no qual ele ocorre e a solução proposta. Os padrões de projeto não apenas oferecem soluções prontas para problemas de projeto, mas também se tornaram um vocabulário amplamente adotado entre desenvolvedores de software. Conhecimento desses padrões pode ser útil tanto ao implementar sistemas próprios quanto ao utilizar sistemas de terceiros, facilitando a compreensão da estrutura e comportamento das classes e componentes. Os padrões de projeto visam criar sistemas flexíveis e extensíveis, antecipando futuras mudanças no sistema para evitar retrabalho no futuro (VALENTE, 2020).

Os padrões de projeto são frequentemente categorizados em três grupos principais: padrões de criação, padrões de estrutura e padrões de comportamento. Os padrões de criação abordam a maneira como os objetos são criados e inicializados, fornecendo soluções para problemas como a criação de objetos de forma flexível e eficiente. Os padrões de estrutura lidam com a composição de classes e objetos para formar estruturas maiores, facilitando a criação de sistemas complexos e mantendo sua flexibilidade. Por fim, os padrões de comportamento tratam da interação entre objetos e classes, fornecendo soluções para problemas como a comunicação entre componentes de um sistema e a distribuição de responsabilidades. Cada categoria de padrões de projeto tem seu papel único na criação de sistemas de software robustos e flexíveis, ajudando os desenvolvedores a resolver uma variedade de problemas de projeto comuns (VALENTE, 2020).

O padrão de projeto *Template Method* é um padrão comportamental que define o esqueleto de um algoritmo em uma classe base, mas permite que subclasses implementem ou substituam partes específicas desse algoritmo sem alterar sua estrutura geral. Isso promove a reutilização de código e permite a customização do comportamento do algoritmo em diferentes contextos (VALENTE, 2020).

2.5 Trabalhos Relacionados

Nesta seção, são reunidos os trabalhos que apresentam *frameworks* para experimentos de sistemas de recomendação, com semelhança ao *framework* original abordado neste estudo. As investigações apresentadas neste contexto estão focadas na área de pesquisa de Sistemas de Recomendação, servindo como recursos valiosos para o desenvolvimento de aplicações nesse domínio. Ao final, na Seção 2.5.6 há uma discussão sobre todos os pontos dos projetos citados abaixo.

2.5.1 RecSysExp

A base para o desenvolvimento deste trabalho é a monografia de Natali (2023), onde nasceu o **RecSysExp**, construído com base em arquiteturas e funcionalidades de outros projetos, como a tese de doutorado de Fortes (2022). Influências significativas vêm de bibliotecas e *frameworks* como *LensKit* (Seção 2.5.4), Elliot (ANELLI et al., 2021), e *Xperimentor* (Seção 2.5.5).

O **RecSysExp** é concebido para ser utilizado por dois atores principais: o desenvolvedor e o experimentador. O caso de uso geral envolve quatro tarefas principais: planejar um experimento, executar um experimento, analisar um experimento e desenvolver novas funcionalidades. Essa divisão busca oferecer praticidade tanto para experimentadores quanto para desenvolvedores. A arquitetura geral é dividida em três partes: pré-processamento, modelagem e treinamento, e avaliação e visualização dos resultados. Cada parte desempenha um papel crucial no *framework*, contribuindo para a realização de experimentos de recomendação.

Na fase de pré-processamento, a base de dados dos sistemas de recomendação passa por uma série de operações, como normalização dos dados, seleção de dados de treino e teste, e codificação de variáveis textuais. Essa etapa desempenha um papel crucial na definição das características fornecidas à fase de modelagem e treinamento, possibilitando o desempenho adequado dos modelos. Na fase de modelagem e treinamento são delineadas as etapas cruciais para obter recomendações dentro do *framework*. Este módulo encapsula todos os algoritmos de recomendação, abrangendo abordagens colaborativas, baseadas em conteúdo, híbridas, popularidade, entre outras. A estrutura flexível é construída com interfaces e classes abstratas, permitindo a implementação de diferentes algoritmos apenas estendendo essas interfaces e classes. A etapa final do *framework* é avaliação e visualização, focando a obtenção de informações cruciais sobre o modelo definido e os itens recomendados. Inicialmente, os resultados são processados para extrair valor máximo dos artefatos gerados, utilizando recursos estatísticos e diversas métricas de avaliação.

2.5.2 RecMetrics

O *RecMetrics* representa um *framework* que realiza o cálculo de *meta-features* em extensas bases de dados para Sistemas de Recomendação, utilizando abordagens de Filtragem Colaborativa e Baseada em Conteúdo. Esta ferramenta, inicialmente desenvolvida por Souza (2014) e aprimorada por Bruckner (2017), baseia-se em uma arquitetura paralela para efetuar esses cálculos.

O processo inicia-se com a criação de uma matriz de avaliações item-usuário a partir de um arquivo de texto contendo todas as avaliações. Essa etapa utiliza a estrutura *DataModel* do *framework Apache Mahout*. O vetor de preferências é processado por múltiplas *threads*, que calculam os resultados e os escrevem concorrentemente em um arquivo de texto no disco.

Apesar de suas capacidades, o *RecMetrics* possui limitações, como a exclusividade no trabalho com dados de Filtragem Colaborativa e a necessidade de configurar cerca de vinte parâmetros, incluindo as *meta-features* desejadas e informações de entrada e saída, tornando seu uso complexo. [Bruckner \(2017\)](#) buscou superar essas limitações ao dividir a arquitetura em três etapas: preparação, processamento e saída dos dados.

Na fase de preparação, o sistema lê as configurações especificadas pelo usuário, incluindo o tipo de entrada de dados e as *meta-features* desejadas. Isso permite a criação das estruturas de dados necessárias para o cálculo e escrita das *meta-features*. A segunda fase envolve o processamento dos recursos gerados na etapa anterior, utilizando um número variável de *threads* para calcular métricas para itens ou usuários, resultando em arquivos contendo os resultados.

A avaliação comparativa indicou que as *meta-features* de Filtragem Colaborativa não apresentaram grandes discrepâncias de desempenho em relação ao *RecMetrics*. Por outro lado, as *meta-features* baseadas em conteúdo mostraram uma significativa melhoria em comparação com o software *CBRecommender*, especialmente devido ao paralelismo durante o cálculo das métricas. Apesar dos resultados, a ferramenta demonstrou avanços notáveis, como facilidade de configuração e execução, comparada ao seu predecessor, a facilidade de implementar novas *meta-features*, e a capacidade de calcular *meta-features* em grandes bases de dados, independentemente da abordagem (colaborativa ou baseada em conteúdo).

2.5.3 LibRec

[Guo et al. \(2015\)](#) apresentaram o *LibRec*, uma biblioteca *open-source* em Java que implementa diversos algoritmos de recomendação, juntamente com um conjunto de métricas de avaliação. O projeto é voltado para abordar dois problemas fundamentais em Sistemas de Recomendação: a predição de avaliações (*ratings*) e a recomendação de itens. A arquitetura do *LibRec*, destaca três elementos principais: interfaces genéricas, estruturas de dados e algoritmos de recomendação.

Dentro das interfaces genéricas, estão definidos conjuntos de recomendações abstratas, como *Recommender* (definindo um recomendador genérico estendido por *baseline* e outros algoritmos), *IterativeRecommender* (geralmente baseado em aprendizado iterativo), *GraphicRecommender* (adequado para modelos gráficos probabilísticos), *SocialRecommender* (incorpora informações sociais) e *ContextRecommender* (integra informações contextuais adicionais, como temporais, na recomendação).

No contexto de Sistemas de Recomendação, as estruturas de dados comumente utilizadas são matrizes e vetores densos ou esparsos. Os autores optaram por utilizar a biblioteca Java MTJ (*Matrix Toolkit Java*) para manipulação de matrizes, visando otimizar o tempo de execução dos recomendadores. Além disso, foram implementadas técnicas de cache e armazenamento de dados para matrizes e vetores densos usando *arrays* bidimensionais, entre outras melhorias. A

parte dos algoritmos de recomendação é dividida em três tipos principais: *baselines* (utilizando pouca informação personalizada), algoritmos principais (última geração, baseados em contexto e interações usuário-item).

O *LibRec* também oferece várias métricas de avaliação, agrupadas em três principais categorias: medidas preditivas baseadas em erro, medidas baseadas em classificação e outras medidas. Dentre as medidas baseadas em erro, destacam-se *Mean Absolute Error* (MAE), *Root Mean Squared Error* (RMSE) e *Mean Percentage Error* (MPE). No grupo de medidas baseadas em classificação, incluem-se *Mean Average Precision* (MAP), *Normalized Discounted Cumulative Gain* (NDCG), *Mean Reciprocal Rank* (MRR), *Area Under The ROC Curve* (AUC), *Precision and Recall*, entre outras. Adicionalmente, o *LibRec* apresenta novas medidas, incluindo uma medida de diversidade baseada em similaridade.

2.5.4 Lenskit

O *LensKit*, inicialmente desenvolvido em Java (EKSTRAND, 2018) e posteriormente adaptado para Python (EKSTRAND, 2020), é um projeto de código aberto com o propósito de fornecer uma estrutura abrangente para processar dados, avaliar e treinar algoritmos em sistemas de recomendação. Desde sua versão em Java, os criadores destacam conquistas como a ampla cobertura de testes para garantir confiabilidade e a modularidade dos algoritmos, permitindo a substituição e reconfiguração de componentes.

Ao longo do desenvolvimento, o *LensKit* for Python estabeleceu objetivos específicos para tornar-se mais útil e acessível. A abordagem escolhida envolve a integração de bibliotecas populares, como *Scikit-Learn*, *PyTorch*, *TensorFlow*, *Pandas* e *PyData*, em vez de criar soluções independentes. Essa estratégia, semelhante ao objetivo deste trabalho, amplia o *LensKit* com API (*Application Programming Interface*) comuns para algoritmos de recomendação, preparação de dados, processamento em lote, além de diversos algoritmos de filtragem colaborativa e métricas de avaliação.

O *LensKit* se diferencia de outros pacotes ao permitir a modificação do pipeline de dados conforme necessário, em vez de ocultá-lo. Destacam-se quatro principais casos de uso, incluindo a avaliação de sistemas de recomendação, pesquisa *off-line*, aplicações educacionais e produção em pequena escala.

Assim, o *LensKit* visa oferecer métricas de avaliação, algoritmos de filtragem colaborativa e funcionalidades para experimentos de recomendação. Integrações específicas do *LensKit* serão incorporadas a este *framework* para operações comuns, como modelos de recomendação e métricas.

2.5.5 Xperimentor

Em um cenário onde a administração de uma ampla variedade de experimentos, cada um com suas particularidades de entrada, é essencial, surge a necessidade de um eficiente gerenciamento para evitar abordagens manuais dispendiosas. De acordo com a pesquisa de [Dias \(2019\)](#), constatou-se a ausência de um sistema de gestão de experimentos que se concentre na modelagem e condução dos mesmos. Os *frameworks* existentes exigem uma configuração prévia e o armazenamento em máquinas virtuais, resultando em uma construção manual que acarreta em problemas como a utilização ineficiente dos recursos computacionais.

São mencionados projetos como *RecMetrics* (Seção 2.5.2), *LensKit* (Seção 2.5.4) e *LibRec* (Seção 2.5.3), cada um apresentando características específicas em relação à linguagem de programação, arquitetura e funcionalidades. O foco do trabalho é a apresentação de um novo *framework* que visa integrar e expandir as capacidades dessas ferramentas existentes, oferecendo flexibilidade, simplicidade e um conjunto abrangente de recursos. É relevante destacar que, além dos elementos comuns aos *frameworks* mencionados, este projeto adiciona a inovação de uma interface gráfica amigável, proporcionando uma abordagem mais acessível e visual para usuários finais.

Portanto, assim como os *frameworks* citados, que se dedicam a diversas facetas dos Sistemas de Recomendação, este trabalho busca ir além, acrescentando a camada visual por meio de uma interface gráfica. Isso não apenas reforça a proposta de flexibilidade e expansão do conjunto de recursos, mas também visa tornar a experiência do usuário mais intuitiva e eficaz na interação com os resultados e funcionalidades do *framework*.

2.5.6 Discussão

Em termos de objetivo geral, todos esses trabalhos estão ligados a Sistemas de Recomendação, buscando inovações e melhorias nesse campo.

Cada subseção, ao mencionar a base teórica e desenvolvimento, ressalta a influência de trabalhos anteriores e *frameworks* existentes. Esse método de construção sobre o conhecimento já existente destaca a evolução contínua dessas ferramentas. A divisão em fases ou estágios é uma característica comum entre eles, com processos bem delineados de preparação, processamento e avaliação. Isso reflete uma abordagem metodológica que visa a eficiência e a clareza nas etapas operacionais.

Outro ponto de convergência é a flexibilidade oferecida por esses *frameworks*. Eles permitem a implementação de diferentes algoritmos e acomodam diversas abordagens de recomendação, visando atender a uma ampla gama de necessidades. Tanto o *RecMetrics* quanto o *LibRec* estão focados no cálculo de *meta-features* para sistemas de recomendação. Enquanto o *RecMetrics* destaca limitações, como a exclusividade no trabalho com dados de Filtragem Colaborativa, o *LibRec* oferece uma abordagem mais ampla com interfaces genéricas e estruturas

de dados específicas.

O *LensKit* se destaca por sua modularidade e testes extensivos, além de sua adaptação para Python, integrando bibliotecas populares e tornando-se mais acessível. Em contrapartida, o *Xperimentor* inova ao adicionar uma camada visual por meio de uma interface gráfica, proporcionando uma experiência mais intuitiva e eficaz para usuários finais.

Esses *frameworks*, ao apresentar suas fases específicas de trabalho, revelam nuances distintas em suas abordagens. Por exemplo, a fase de preparação no *RecMetrics*, as interfaces genéricas e estruturas de dados no *LibRec*, a modularidade e a integração de bibliotecas no *LensKit*, e a inovação de uma interface gráfica no *Xperimentor*. Cada *framework* destaca limitações e melhorias, com o *RecMetrics* reconhecendo suas restrições e buscando superá-las, enquanto o *LensKit* destaca suas conquistas, como a ampla cobertura de testes para garantir confiabilidade.

As métricas de avaliação apresentadas por esses *frameworks* variam, refletindo diferentes preocupações e abordagens para avaliar o desempenho dos sistemas de recomendação. Em resumo, cada capítulo contribui para o desenvolvimento contínuo dos Sistemas de Recomendação, abordando diferentes facetas e introduzindo inovações específicas para atender às demandas em constante evolução nesse campo.

3 Desenvolvimento

Este capítulo aborda, em detalhes, o processo prático de execução do projeto proposto. Ao longo deste capítulo, serão apresentadas as estratégias, métodos e implementações adotadas para atingir os objetivos traçados. Cada fase do desenvolvimento é detalhada, proporcionando uma compreensão das decisões tomadas, dos desafios enfrentados e das soluções encontradas. As tecnologias que aqui serão citadas podem ser estudadas no Capítulo 2

As atividades essenciais para atingir o objetivo estabelecido envolveram a criação de uma SPA capaz de configurar todos os módulos do *framework* (Seção 3.1). Além disso, buscou-se encapsular todas as dependências do *framework* por meio da criação de uma imagem *Docker* (Seção 3.2). Adicionalmente, foi realizada uma modificação direta no código-fonte do *framework*, abrangendo uma alteração na arquitetura de classes relacionada aos *datasets* (Seção 3.3).

Todos os códigos fonte, manuais e informações adicionais referentes ao sistema desenvolvido, estão disponíveis em um repositório no GitHub. O acesso ao repositório pode ser feito através do seguinte link: <<https://github.com/SanCunha/recsysexp/tree/master>>.

3.1 Criação da Interface Gráfica

Para o desenvolvimento da SPA (veja Subseção 2.3) foi utilizado *React* (veja Subseção 2.3.1) e *Next.js* (veja Subseção 2.3.2). A linguagem de programação escolhida foi o *Typescript*, por isso, para cada módulo presente no arquivo de configuração existe uma interface para que seja possível armazenar e alterar seus estados. O componente principal da aplicação contém um estado que é um objeto adaptado do arquivo de configuração inicial. Esse mesmo componente é responsável por passar propriedades e funções de alteração para componentes filhos.

A escolha do *Next.js* foi motivada pela sua eficácia na configuração inicial do *React*, simplificando a estruturação de pastas e a criação de arquivos padrões. *Next.js* oferece uma abordagem de carregamento sob demanda (*lazy loading*), o que significa que os componentes são carregados apenas quando necessários, melhorando significativamente o tempo de carregamento da página. Essa característica é crucial para aprimorar a experiência do usuário, especialmente em aplicações complexas, onde o carregamento rápido de conteúdo é uma prioridade.

A construção da interface é pautada pelos módulos do *framework* que podem ser configurados via arquivo inicial. Esses módulos e seus respectivos algoritmos foram colocados em um arquivo JSON que serve de auxílio na criação dos campos de configuração. Por ser um arquivo muito verboso não convém mostrá-lo aqui, o mesmo pode ser conferido em <<https://github.com/SanCunha/recsysexp/blob/master/backend/config-full.json>>. Nesse primeiro momento, caso o usuário adicione novas classes ao *framework* é necessária a atualização manual

desse arquivo. A automatização desse processo é umas das propostas para trabalhos futuros citada na Seção 5.2.

As figuras a seguir mostram a página inicial do sistema (Figura 3.1) que mostra os campos expansíveis de configuração e o botão *RUN* responsável por enviar a requisição ao servidor para que o experimento seja executado, em seguida temos os campos configuráveis de cada módulo. (Figuras 3.2 à 3.6).

Figura 3.1 – Tela inicial.



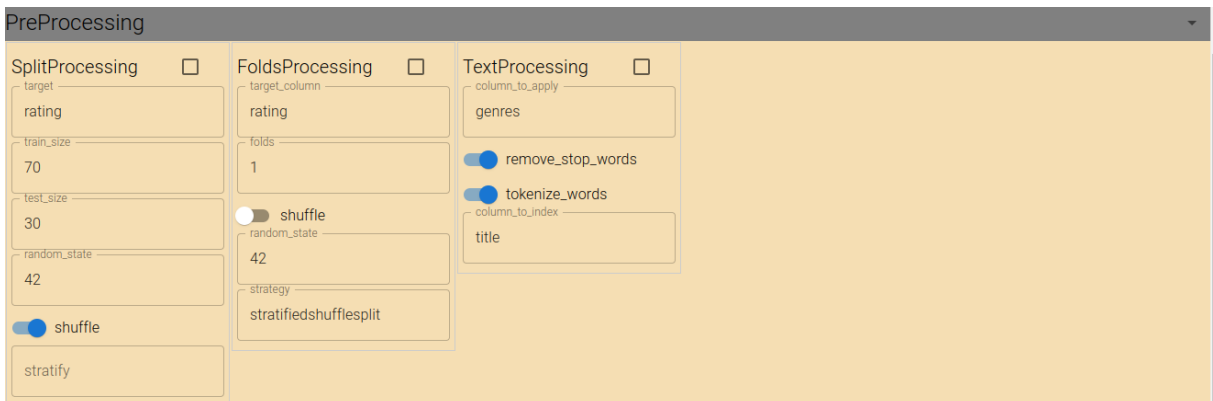
Fonte: Elaborado pelo autor (2024)

Figura 3.2 – Campos de Configuração de *Dataset*.



Fonte: Elaborado pelo autor (2024)

Figura 3.3 – Campos de Configuração de *Preprocessing*.



Fonte: Elaborado pelo autor (2024)

Figura 3.4 – Campos de Configuração de *Metrics*.

Fonte: Elaborado pelo autor (2024)

Figura 3.5 – Campos de Configuração de *Recommenders*.

Fonte: Elaborado pelo autor (2024)

Figura 3.6 – Campos de Configuração de *Visualization*.

Fonte: Elaborado pelo autor (2024)

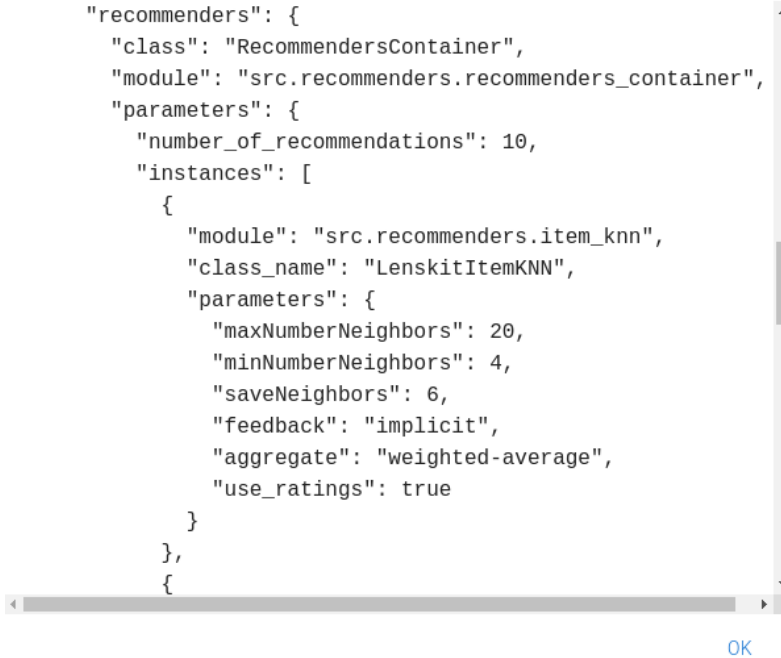
Nas Figuras de configuração é possível observar que existe um *checkbox* para cada módulo que se deseja acrescentar ao experimento, ao marcar, indica-se que fará parte da execução. No final da página existe um botão para que seja possível executar o experimento com as configurações escolhidas, antes de fazer a requisição na máquina que está hospedando o *framework* é mostrado para o usuário as informações escolhidas para que seja confirmado (Figura 3.7), caso sim, inicia-se o processo do lado do servidor para retornar os resultados para o usuário. Ao final da execução é retornado os arquivos gerados pelo **RecSysExp**.

Figura 3.7 – Visualização do experimento montado

```
Config File

"recommenders": {
  "class": "RecommendersContainer",
  "module": "src.recommenders.recommenders_container",
  "parameters": {
    "number_of_recommendations": 10,
    "instances": [
      {
        "module": "src.recommenders.item_knn",
        "class_name": "LenskitItemKNN",
        "parameters": {
          "maxNumberNeighbors": 20,
          "minNumberNeighbors": 4,
          "saveNeighbors": 6,
          "feedback": "implicit",
          "aggregate": "weighted-average",
          "use_ratings": true
        }
      }
    ],
  },
  {

```



Fonte: Elaborado pelo autor (2024)

Por fim, é necessário ressaltar as comunicações realizadas com o servidor (Seção 3.2). Três requisições são realizadas para que a execução aconteça, duas envolvem *datasets*, a primeira é feita ao carregar a página inicial para verificar quais *datasets* já foram inseridos e povoar o *select* mostrado na Figura 3.2. Caso o usuário deseje fazer o *upload* de um novo *dataset* é realizada a requisição para armazenar o novo arquivo.

3.2 Encapsulamento usando *Docker*

De maneira resumida, o *Docker* (veja Subseção 2.2) pode ser dividido em duas partes, imagem e contêiner. A imagem carrega todas as informações necessárias para o ambiente que será executado dentro do contêiner. O *Dockerfile* contém uma série de instruções executadas sequencialmente para a criação de um ambiente de execução. O arquivo final é mostrado no Algoritmo 3.1.

A base da imagem é o sistema operacional Ubuntu, com a versão mais recente selecionada como ponto de partida (Linha 1). Para garantir a não interatividade durante a instalação de pacotes, é definida uma variável de argumento chamada *DEBIAN_FRONTEND* (Linha 2).

O diretório de trabalho dentro do contêiner é configurado como */app* (Linha 3), indicando que as operações subsequentes serão realizadas neste local. São então executadas as atualizações dos repositórios de pacotes e a instalação do pacote *software-properties-common* (Linhas 4),

Algoritmo 3.1: Dockerfile RecSysExp

```
1 FROM ubuntu:latest
2 ARG DEBIAN_FRONTEND=noninteractive
3 WORKDIR /app
4 RUN apt-get update -y && apt-get install -y software-properties
   -common
5 RUN add-apt-repository ppa:deadsnakes/ppa -y && apt-get update
   -y
6 RUN apt-get install -y python3.8 python3.8-venv python3.8-dev
   python3-pip
7 RUN python3.8 -m venv myenv
8 SHELL ["/bin/bash", "-c"]
9 RUN source myenv/bin/activate
10 RUN pip install --upgrade pip
11 COPY requirements.txt /app
12 RUN pip install -r requirements.txt
13 RUN pip install --ignore-installed Flask
14 RUN pip install flask-cors
15 RUN apt-get install -y gcc
16 RUN apt-get install -y openjdk-11-jre-headless
17 COPY . /app
18 EXPOSE 5000
19 CMD ["python3", "app.py"]
```

necessário para utilizar o comando *add-apt-repository* (Linha 5).

As próximas etapas concentram-se na configuração do ambiente *Python*. O *Python* 3.8 é instalado (Linha 6), um ambiente virtual chamado *myenv* é criado (Linha 7), e o *shell* é configurado para o modo *Bash* (Linha 8). *myenv* é iniciado (Linha 9), O *pip* é atualizado para a versão mais recente (Linha 10), e as dependências do *Python* especificadas no arquivo *requirements.txt* são instaladas no ambiente virtual (Linhas 11 e 12).

Adicionalmente, as extensões *Flask* e *Flask-CORS* são instaladas (Linhas 13 e 14) para suportar o desenvolvimento da aplicação. O contêiner é equipado com o compilador GCC e o OpenJDK 11 (Linhas 15 e 16), adequados para compilação e execução de códigos em C e em Java.

O conteúdo do diretório local do *Docker*, onde o *Dockerfile* está localizado, é copiado para o diretório */app* dentro do contêiner (Linha 17). A porta 5000 é exposta para permitir a comunicação externa com a aplicação (Linha 18). Ao iniciar o contêiner, o comando padrão especifica a execução do *script Python* *app.py*, iniciando assim a aplicação (Linha 19).

Em síntese, o *Dockerfile* proporciona um ambiente robusto e bem-configurado para a execução da aplicação *Python*, incluindo todas as dependências necessárias e configurações específicas para o contexto do projeto.

Um ponto que precisa ser destacado em relação a composição do *Dockerfile*, é a necessidade de um sistema operacional, como o *framework* executa algoritmos em diversas linguagens é necessário uma base para essa execução. Também é possível verificar que o código base é implementado usando *Python*. A necessidade do *Flask* é explicada abaixo.

Como não é um objetivo desse trabalho, a adição de uma camada web no *framework* não terá um grande detalhamento. Para evoluir o *framework* que originalmente operava localmente, o **RecSysExp** foi transformado em um servidor. A motivação para essa transição surgiu da necessidade de permitir que o *framework* atendesse a requisições externas da nossa UI, ampliando seu alcance e utilidade. O *Flask* é um *framework web* em *Python*, e foi escolhido devido à sua simplicidade e flexibilidade. Durante o processo de transformação, foram mantidas as características essenciais do *framework* original, assegurando que suas funcionalidades principais permanecessem inalteradas. Com a adição das capacidades do *Flask*, agora *framework* é capaz não apenas de processar dados localmente, mas também de receber requisições externas via HTTP e retornar respostas apropriadas.

O mesmo processo foi realizado para o *Dockerfile* responsável pela criação da imagem da UI. Como podemos observar abaixo, esse é um arquivo de configurações mais simples.

Algoritmo 3.2: *Dockerfile* UI

```
1 FROM node:latest
2 WORKDIR /app
3 COPY package*.json ./
4 COPY next.config.js ./
5 RUN npm install -g next@13.4.9
6 RUN npm install
7 COPY . .
8 EXPOSE 3000
9 CMD [ "npm" , "run" , "dev" ]
```

Partindo a última versão do node (Linha 1), o diretório de trabalho dentro do contêiner é configurado como /app (Linha 2), indicando que as operações ocorrerão neste local. Os arquivos cruciais *package*.json* e *next.config.js* são copiados do diretório de construção do *Docker* para o diretório /app no contêiner, estabelecendo as configurações e dependências necessárias para a aplicação *Next.js* (Linhas 3 e 4).

A imagem é configurada para instalar globalmente a versão específica 13.4.9 do *framework* *Next.js*, garantindo consistência na execução da aplicação (Linha 5). As dependências do projeto são instaladas através do comando *npm install* (Linha 6), conforme especificado no arquivo *package.json*.

Todo o conteúdo do diretório de construção do *Docker* é copiado para /app no contêiner, incluindo todos os arquivos da aplicação (Linha 7). A porta 3000 é exposta, permitindo a

comunicação externa com a aplicação (Linha 8). O comando padrão especificado para iniciar o contêiner é `npm run dev`, utilizado para iniciar um servidor de desenvolvimento em ambientes *Next.js* (Linha 9).

Resumidamente, o *Dockerfile* cria uma imagem configurada para executar uma aplicação *Next.js*, garantindo a adequada instalação de dependências, configurações e exposição de portas. Este processo é fundamental para garantir a eficiente execução da aplicação em um ambiente de contêiner.

Após a criação dos dois *Dockerfiles* temos as imagens necessárias para executar nossos dois sistemas dentro de contêineres. Nesse ponto é possível a execução individual de cada imagem através do comando `docker build -t nome-da-imagem .` na pasta onde se encontra o *Dockerfile*.

Este é o comando principal utilizado para iniciar o processo de construção de uma imagem *Docker*. A opção `-t` (ou `-tag`) é usada para atribuir um nome ou uma “tag” à imagem que está sendo construída. O nome-da-imagem é o identificador que será usado para referenciar essa imagem posteriormente. A *tag* pode ser, por exemplo, uma versão ou uma descrição adicional. O ponto (.) representa o contexto do *build*. Este é o diretório do sistema de arquivos que será usado como contexto para a construção da imagem. Todos os arquivos e subdiretórios neste diretório serão considerados no processo de construção. Para executar uma imagem individual o comando é `docker run nome-da-imagem`.

Para que duas ou mais imagens sejam instanciadas em conjunto é necessário a criação de um arquivo chamado *docker-compose* que tem o papel e configuração similar ao *Dockerfile*. Na pasta raiz do projeto se encontra o Algoritmo 3.3.

Neste arquivo, a versão 3 é especificada para garantir a compatibilidade e utilização dos recursos mais recentes.

Dois serviços principais são definidos no contexto deste arquivo: “*flask-app*” e “*react-app*”. O primeiro refere-se a um contêiner destinado a hospedar uma aplicação Flask (Nosso *framework*), enquanto o segundo é destinado a um aplicativo React (Nossa UI). Cada serviço é configurado para construir o respectivo contêiner a partir do código-fonte localizado nos diretórios “*backend*” e “*frontend*”, utilizando os arquivos *Dockerfile* presentes nesses diretórios.

O serviço “*flask-app*” é configurado para mapear a porta 5000 do contêiner para a porta 5000 do *host*, permitindo o acesso à aplicação *Flask*. Além disso, é estabelecido um volume compartilhado entre o *host* e o contêiner, mapeando o diretório “*backend*” para o diretório “*/app*” no contêiner.

O serviço “*react-app*” segue uma abordagem semelhante, com a diferença de que mapeia a porta 3000 do contêiner para a porta 3000 do *host*. Além disso, ele também estabelece uma dependência em relação ao serviço “*flask-app*”, garantindo que o serviço *Flask* seja iniciado antes do serviço *React*.

Algoritmo 3.3: *docker-compose*

```
1 version: "3"
2
3 services:
4   flask-app:
5     build:
6       context: ./backend/
7       dockerfile: Dockerfile
8     ports:
9       - "5000:5000"
10    volumes:
11      - ./backend:/app
12
13
14   react-app:
15     build:
16       context: ./frontend/
17       dockerfile: Dockerfile
18     ports:
19       - "3000:3000"
20    volumes:
21      - ./frontend:/app
22    depends_on:
23      - flask-app
```

Como todas as configurações necessárias já estão contidas nos arquivos de criação das imagens, para executar todo o sistema é necessário apenas dois comandos na pasta onde se encontra o arquivo *docker-compose.yml*. O primeiro ***docker-compose build*** responsável pela criação das imagens. Esse comando executa os passos descritos no *docker-compose* e pode levar alguns minutos. Após a criação das imagens o comando ***docker-compose run*** instancia os contêineres.

Existe duas formas de verificar se tudo ocorreu como deveria. A primeira é acessando <https://localhost:3000> onde deverá aparecer a Figura 3.1. A segunda é acessando <https://localhost:5000/status> onde é indicado se o servidor está online.

3.3 Alteração na Arquitetura da Classe *Datasets*

A interface *Dataset* fornece métodos que representarão as estruturas de *ratings*, *users* e *items*. Cada um desses métodos retornará valores correspondentes ao contexto da base de dados que será instanciada, ou seja, caso esse conjunto de dados diga respeito a filmes, pense que os *ratings* serão representados por uma matriz de avaliações feitas por usuários para determinados filmes, os *users* como uma matriz onde cada usuário possuirá um conjunto de informações como

idade e sexo, por exemplo (NATALI, 2023).

O objeto de configuração relacionado a configuração da classe *dataset* da versão original pode ser conferido abaixo (Algoritmo 3.4).

Algoritmo 3.4: Objeto de configuração do *framework* original

```
{
  "dataset": {
    "class": "MovieLens",
    "module": "src.data.movielens",
    "parameters": {
      "proportion": "ml-latest-small",
      "filters": {}
    }
  }
}
```

Como pode ser reparado, nessa configuração é necessário ter dentro do *framework* um módulo do *dataset* desejado. Para solucionar esse problema o novo objeto de configuração é mostrado abaixo.

Algoritmo 3.5: Classe *GeneralDataset*

```
class GeneralDataset(ABC):
    def __init__(self, parameters: dict):
        folder = "data_storage/" + parameters["name"]
        + "/items.csv"
        self.dataset = parameters["name"]
        self.origin = os.path.join(os.getcwd(), folder)
        self.destiny_folder = os.path.join(os.getcwd(), "
            experiment_output/datasets")

    def run(self):
        self.new_feature()
        self.run_datasets(self.origin, self.destiny_folder)

    def run_datasets(self, origin, destiny):...

    @abstractmethod
    def new_feature(self):
        pass
```

A classe criada (Algoritmo 3.5) representa uma base geral projetada para ser utilizada quando os arquivos seguem o padrão estabelecido pelo *framework*, padrão esse que é um arquivo .csv com colunas item, user e rating nessa ordem. Essa abordagem padronizada facilita a consistência e a interoperabilidade entre diferentes conjuntos de dados e sistemas.

No entanto, é reconhecido que cada conjunto de dados pode apresentar suas próprias peculiaridades e exigências específicas. Para lidar com essas situações, é possível estender a classe base existente, criando uma nova classe que herda suas funcionalidades. Isso permite a implementação de lógicas adicionais ou modificações específicas para lidar com as particularidades de um determinado conjunto de dados, mantendo ao mesmo tempo a estrutura geral e a coesão do código. Essa abordagem de herança e extensibilidade possibilita adaptar a solução de forma flexível para atender às necessidades específicas de cada conjunto de dados, garantindo assim sua eficácia e versatilidade em diferentes contextos.

4 Resultados

Neste capítulo, serão apresentados em detalhes todos os procedimentos realizados desde a instalação até a execução do *framework*. Além disso, será discutida a configuração necessária para replicar o experimento descrito na monografia de Natali (2023). Este experimento tem como objetivo validar os recursos apresentados no capítulo anterior, fornecendo uma avaliação prática de sua funcionalidade e eficácia, além de uma visão completa do processo de execução do experimento e dos resultados alcançados, contribuindo para a validação e compreensão do funcionamento do *framework* em um contexto prático. Os resultados obtidos são discutidos proporcionando uma análise e uma compreensão do desempenho do *framework* em relação ao experimento replicado.

Aqui está demonstrado passo a passo tudo que foi necessário para replicar o experimento utilizado como teste da primeira versão do *framework* e a comparação dos resultados obtidos. Iniciando com a criação das imagens necessárias Seção 4.1, seguido pela inicialização do sistema Seção 4.2. Avançando para o processo de *Upload* do arquivo de dados, exemplificado com o conjunto *MovieLens* na Seção 4.3. Em seguida, a configuração do experimento é mostrada na Seção 4.4 e por fim a comparação dos resultados na Seção 4.5.

4.1 Criação das imagens

Na pasta raiz do diretório (pasta onde se encontra o arquivo *docker-compose.yaml*) foi executado o comando *docker-compose build* para que as imagens fossem construídas. Esse passo demora alguns minutos, mas precisa ser executado apenas na primeira vez.

4.2 Inicialização do sistema

Na mesma pasta citada na seção anterior, o comando *docker-compose run* inicializou o sistema. As duas condições de sistema funcionando citadas na Seção 3.2 foram atendidas. Obtendo a Figura 3.1 como confirmação da UI e a Figura 4.1 como confirmação do servidor.

Figura 4.1 – Resposta do Servidor Sobre *Status*

```
1 // 20240205114534
2 // http://localhost:5000/status
3
4 {
5   "Online": true
6 }
```

Fonte: Elaborado pelo autor (2024)

4.3 Upload do arquivo

O arquivo utilizado é *ratings.csv* presente no conjunto de dados do *MovieLens*, utilizando a proporção *ml-latest-small*. Como pode ser reparado na Figura 4.2 o arquivo é renomeado para *MovieLens.csv* pois esse será o nome que será salvo dentro do *framework*. Ao clicar em *Upload* o arquivo é enviado para o servidor e caso o usuário não decida usar um outro *dataset* já existente ficará salvo como a escolha final.

Figura 4.2 – Upload *MovieLens*

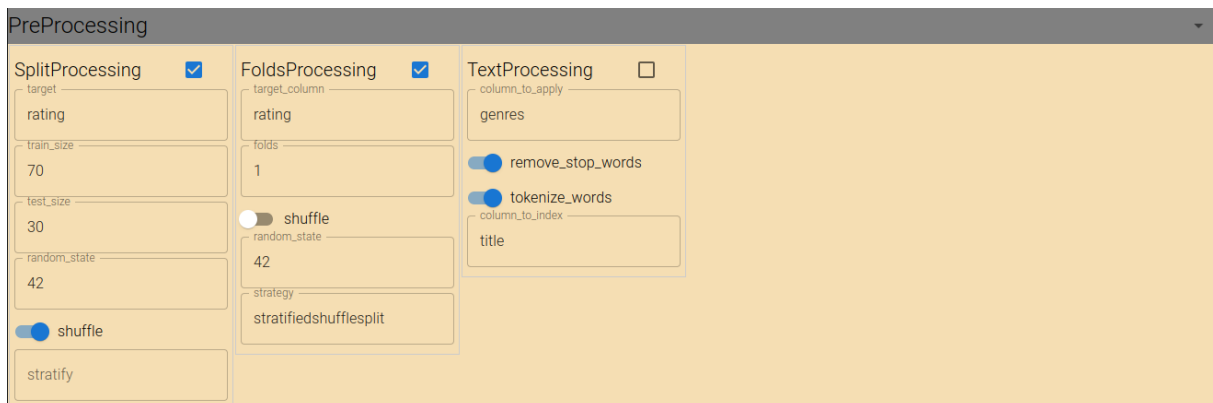


Fonte: Elaborado pelo autor (2024)

4.4 Configuração do Experimento

A configuração do *dataset* foi realizada ao fazer o *Upload* na Subseção 4.3. As configurações de *Preprocessing* e *Visualization* não são especificadas por Natali (2023), mas, pela estrutura do experimento e pelas imagens resultantes no trabalho base, elas ficam explícitas como mostrado nas Figuras 4.3 e 4.6.

Figura 4.3 – Configuração de *Preprocessing*

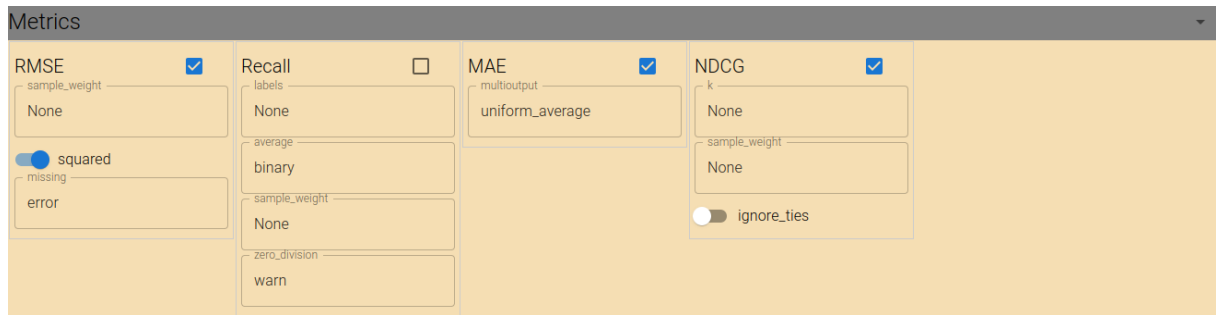


Fonte: Elaborado pelo autor (2024)

4.5 Comparação dos Resultados

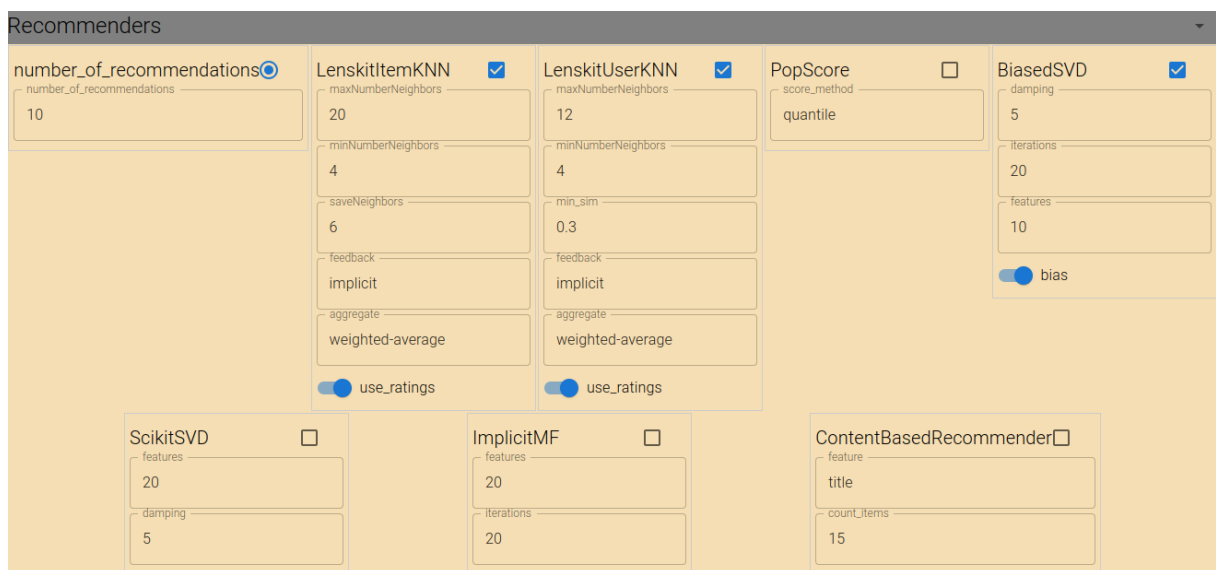
O intuito desses experimentos não é fazer uma análise considerando os valores gerados para cada algoritmo, mas sim, atestar que o *framework* é capaz de realizar tal processo. Da mesma maneira, serão consideradas as visualizações geradas, sendo que o objetivo não é evidenciar todas as possibilidades, mas sim, mostrar como essas imagens serão visualizadas após o término da execução do experimento (NATALI, 2023).

Figura 4.4 – Configuração de *Metrics*



Fonte: Elaborado pelo autor (2024)

Figura 4.5 – Configuração de *Recommenders*



Fonte: Elaborado pelo autor (2024)

Figura 4.6 – Configuração de *Visualization*



Fonte: Elaborado pelo autor (2024)

A Tabela 4.1 mostra os resultados obtidos por este trabalho (v2.0) em relação aos resultados obtidos na primeira versão (Original) do *framework*. Os resultados obtidos neste estudo demonstram uma proximidade notável com os valores do experimento base. Essa consistência entre os resultados reforça a eficácia das modificações e melhorias implementadas no *framework*. A similaridade dos resultados indica que as alterações realizadas não comprometeram a integridade ou o desempenho do sistema.

As Figuras 4.7 e 4.8 mostram informações obtidas após a execução do experimento.

Respectivamente, a primeira mostra a distribuição de *ratings* por usuário, dando a oportunidade de visualizar que a grande maioria dos usuários avaliam poucos itens. A segunda mostra quais itens são mais avaliados, mostrando que os primeiros itens têm uma taxa muito maior que os últimos.

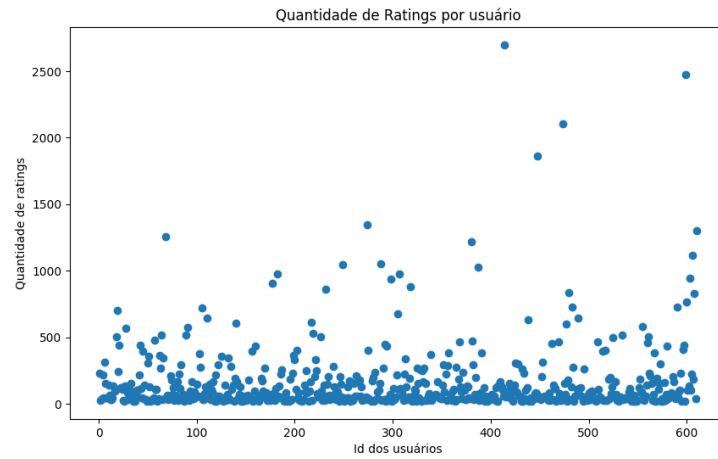
| Experimento - MovieLens | | | | | | |
|-------------------------|----------|----------|----------|----------|----------|----------|
| Recomendadores | RMSE | | NDCG | | MAE | |
| | Original | v2.0 | Original | v2.0 | Original | v2.0 |
| ItemKNN | 1.458033 | 1.250136 | 0.058862 | 0.228218 | 1.161295 | 0.983740 |
| UserKNN | 1.375486 | 1.314336 | 0.062897 | 0.064329 | 1.109742 | 1.039159 |
| BiasedSVD | 1.360211 | 1.279626 | 0.097014 | 0.097014 | 1.102076 | 1.010083 |

Tabela 4.1 – Comparação de resultados obtidos.

O RMSE (Root Mean Square Error) é uma métrica comum para avaliar a precisão de modelos de regressão ou previsão. Ele mede a diferença entre os valores previstos por um modelo e os valores reais dos dados. O RMSE é calculado pela raiz quadrada da média dos quadrados das diferenças entre os valores previstos e os valores reais. Quanto menor o valor do RMSE, mais preciso é o modelo. O NDCG (Normalized Discounted Cumulative Gain) é uma métrica usada para avaliar a qualidade das classificações em problemas de recomendação ou busca de informações. Ele leva em consideração a relevância e a posição dos itens recomendados ou retornados em uma lista classificada. O NDCG normaliza o DCG (Discounted Cumulative Gain) pela classificação ideal, o que leva em conta a relevância dos itens. Quanto maior o valor do NDCG, melhor é a qualidade da classificação. O MAE (Mean Absolute Error) é uma métrica comum para avaliar a precisão de modelos de regressão. Ele mede a média das diferenças absolutas entre os valores previstos por um modelo e os valores reais dos dados. O MAE fornece uma medida da magnitude média dos erros de previsão, sem considerar sua direção. Quanto menor o valor do MAE, mais preciso é o modelo.

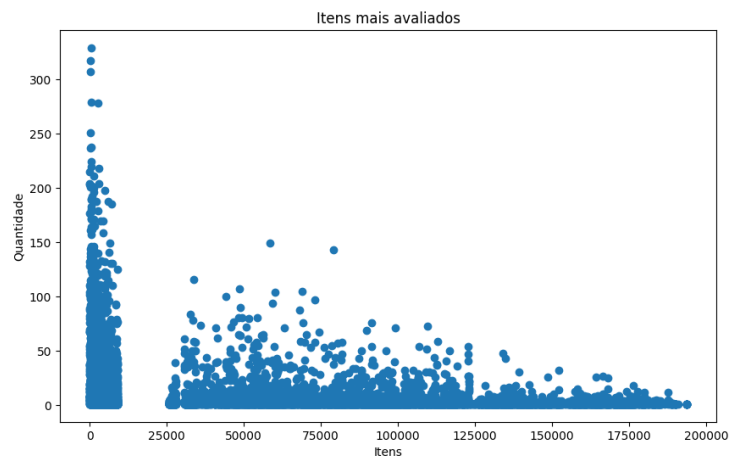
Nosso objetivo aqui não é avaliar a corretude dos algoritmos, mas sim o funcionamento do *framework* com as alterações realizadas. Podemos destacar os seguintes pontos: em primeiro lugar, a facilidade para executar o *RecSysExp* na máquina do usuário aumentou consideravelmente, uma vez que agora a única dependência é o *Docker*, eliminando a necessidade do usuário instalar todas as extensões necessárias para a execução. Em segundo lugar, a configuração do experimento pode ser feita de maneira visual e com alguns cliques, ao invés do preenchimento de um complexo arquivo JSON. Por fim, os resultados obtidos com a replicação do experimento original mostram-se bastante próximos entre si, o que indica que, mesmo com todas as alterações, os elementos principais permanecem funcionais. Para reforçar esse ponto, as Figuras 4.7 e 4.8 são exatamente iguais às encontradas por [Natali \(2023\)](#).

Figura 4.7 – *Ratings* por Usuário



Fonte: Elaborado pelo autor (2024)

Figura 4.8 – Itens mais avaliados



Fonte: Elaborado pelo autor (2024)

5 Considerações Finais

Esse capítulo consolida os principais resultados e ideias obtidas ao longo deste estudo. Aqui, serão apresentadas as conclusões (Seção 5.1) derivadas da análise das melhorias implementadas no **RecSysExp**, destacando não apenas os êxitos alcançados, mas também as lições aprendidas e as áreas que demandam atenção. Em seguida, este capítulo reserva espaço para a exposição de sugestões de trabalhos futuros (Seção 5.2), mostrando possíveis direções de pesquisa que surgem a partir das lacunas identificadas durante esta pesquisa. A partir dessas sugestões, espera-se inspirar e orientar pesquisadores e profissionais interessados em expandir e aprofundar os conhecimentos e avanços alcançados neste estudo.

5.1 Conclusão

Este trabalho surge para aprimorar o **RecSysExp**, abordando questões relacionadas à sua instalação, execução e usabilidade. Foram implementadas melhorias conforme sugerido pelo próprio autor do *framework*. Essas melhorias incluíram a criação de uma interface gráfica intuitiva e acessível, aliada à containerização do sistema, juntamente com uma versão otimizada do *framework* que incorpora uma classe de *datasets* modificada e operacional como uma API.

Ao longo deste trabalho, foi alcançado com sucesso os objetivos propostos, que visavam facilitar o processo de instalação, tornar a configuração do experimento mais visual e didática, além de possibilitar a inclusão de novos *datasets* no sistema de forma simplificada. Essas conquistas não apenas cumprem os desafios iniciais enfrentados, mas também promovem uma experiência mais fluida e eficaz para os usuários do **RecSysExp**.

O desfecho deste trabalho confirma que as melhorias propostas e implementadas foram benéficas, resultando em uma versão aprimorada da ferramenta em foco. Entretanto, é válido destacar que a versão original do *framework* conta com diversos módulos e algoritmos implementados, porém, esses são utilizáveis apenas como bibliotecas em outros projetos. Essa abordagem impõe limitações às ações possíveis por meio da interface gráfica, uma vez que tais módulos e algoritmos não podem ser configurados e executados através de um arquivo de configuração. Diante desse cenário, torna-se evidente que essa característica do sistema restringe a capacidade de configuração por meio da UI.

5.2 Trabalhos Futuros

Com o estudo do código fonte do *framework* e com os resultados obtidos com as novas melhorias serão citados a seguir pontos que podem melhorar significativamente a usabilidade e a

eficiência do **RecSysExp**.

Em primeiro lugar tornar todos os módulos e algoritmos implementados passíveis de configurações via arquivo inicial, essa mudança tornaria possível a criação de experimentos muito mais complexos que os atuais. Com a possibilidade de criar experimentos mais complexos vem a necessidade de poder criar mais de um experimento em uma única configuração, bem como possibilitar a execução replicada de tais experimentos, a interface desenvolvida já contém o suporte necessário para configurar mais de um experimento bastando apenas a adequação do *backend*.

Além de telas de carregamento enquanto espera uma resposta do servidor, a adição de validação de entrada via UI é uma melhoria que impacta diretamente na usabilidade, pois, validando as entradas que o *framework* irá receber a incidências de possíveis erros é menor. Juntamente com isso, adicionar a possibilidade do envio de arquivos de diferentes extensões aumenta as possibilidades de inserção de novos *datasets*. Para conferir o valor gerado por essas modificações é interessante a aplicação de métricas de IHC (Interface Humano-Computador). Métricas como tempo de conclusão de tarefas, taxa de erro, eficiência do sistema, facilidade de aprendizado, satisfação do usuário e engajamento. Além disso, técnicas como testes de usabilidade, análise heurística, questionários de satisfação e observação do comportamento do usuário em situações reais de uso. Essas métricas e técnicas fornecem informações valiosas, permitindo o aprimoramento contínuo das interfaces para oferecer uma experiência mais satisfatória e eficaz aos usuários.

Aplicar o *Template Method* nas outras classes implicará na possibilidade do usuário poder incrementar o *framework* de maneira muito mais simples. Uma maneira de aumentar a velocidade de resposta do sistema é implementando uma paralelização para que as etapas do experimento que não dependem uma da outra possam ser executadas ao mesmo tempo.

Referências

- ALAMDARI, P. M.; NAVIMIPOUR, N. J.; HOSSEINZADEH, M.; SAFAEI, A. A.; DARWESH, A. A systematic study on the recommender systems in the e-commerce. *Ieee Access*, IEEE, v. 8, p. 115694–115716, 2020.
- ANELLI, V. W.; BELLOGÍN, A.; FERRARA, A.; MALITESTA, D.; MERRA, F. A.; POMO, C.; DONINI, F. M.; NOIA, T. D. Elliot: A comprehensive and rigorous framework for reproducible recommender systems evaluation. In: *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. [S.l.: s.n.], 2021. p. 2405–2414.
- BOBADILLA, J.; ORTEGA, F.; HERNANDO, A.; GUTIÉRREZ, A. Recommender systems survey. *Knowledge-based systems*, Elsevier, v. 46, p. 109–132, 2013.
- BRUCKNER, C. H. P. *Framework para extração de meta-features para bases de dados de sistemas de recomendação*. 45 p. Monografia — Universidade Federal de Ouro Preto, 2017. Monografia.
- BURKE, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, Springer, v. 12, p. 331–370, 2002.
- DEVELOPERS, G. *Machine Learning - Recommendation Systems*. 2022. Acesso em: 22 Janeiro 2024. Disponível em: <<https://developers.google.com/machine-learning/recommendation>>.
- DIAS, M. D. P. G. Xperimentor: um framework para o gerenciamento de execução de experimentos computacionais. 2019.
- EKSTRAND, M. D. The lkpy package for recommender systems experiments. 2018.
- EKSTRAND, M. D. Lenskit for python: Next-generation software for recommender systems experiments. In: *Proceedings of the 29th ACM international conference on information & knowledge management*. [S.l.: s.n.], 2020. p. 2999–3006.
- FORTES, R. S. *Aprimorando a recomendação multi-objetivo sobre três novas perspectivas: caracterização dos dados de entrada, sensibilidade ao risco e priorização dos objetivos*. Tese (Tese de Doutorado) — Universidade Federal de Ouro Preto, 2022. Unpublished thesis.
- GUO, G.; ZHANG, J.; SUN, Z.; YORKE-SMITH, N. Librec: A java library for recommender systems. In: CITESEER. *UMAP workshops*. [S.l.], 2015. v. 4, p. 38–45.
- ISINKAYE, F. O.; FOLAJIMI, Y. O.; OJOKOH, B. A. Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*, Elsevier, v. 16, n. 3, p. 261–273, 2015.
- JACKSON, M. React: A javascript library for building user interfaces. *React Documentation*, 2021. Disponível em: <<https://reactjs.org/>>.
- JAVED, U.; SHAUKAT, K.; HAMEED, I. A.; IQBAL, F.; ALAM, T. M.; LUO, S. A review of content-based and context-based recommendation systems. *International Journal of Emerging Technologies in Learning (iJET)*, International Journal of Emerging Technology in Learning, Kassel, Germany, v. 16, n. 3, p. 274–306, 2021. Retrieved January 22, 2024. Disponível em: <<https://www.learntechlib.org/p/219036/>>.

- JOHNSON, R. E. Frameworks=(components+ patterns). *Communications of the ACM*, ACM New York, NY, USA, v. 40, n. 10, p. 39–42, 1997.
- KOREN, Y.; BELL, R.; VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer*, IEEE, v. 42, n. 8, p. 30–37, 2009.
- LÜ, L.; MEDO, M.; YEUNG, C. H.; ZHANG, Y.-C.; ZHANG, Z.-K.; ZHOU, T. Recommender systems. *Physics reports*, Elsevier, v. 519, n. 1, p. 1–49, 2012.
- MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, v. 2014, n. 239, p. 2, 2014.
- METEREN, R. V.; SOMEREN, M. V. Using content-based filtering for recommendation. In: BARCELONA. *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*. [S.l.], 2000. v. 30, p. 47–56.
- NATALI, L. N. M. Monografia, *RecSysExp: um framework de alto nível de abstração para implementação e validação de sistemas de recomendação*. 2023. 82 p. Graduação em Ciência da Computação.
- SCHAFER, J. B.; FRANKOWSKI, D.; HERLOCKER, J.; SEN, S. Collaborative filtering recommender systems. In: *The adaptive web: methods and strategies of web personalization*. [S.l.]: Springer, 2007. p. 291–324.
- SMITH, J.; JOHNSON, A. Single page application: A comprehensive overview. *Journal of Web Development*, v. 15, n. 3, p. 123–135, 2019. Disponível em: <<https://www.example.com/article123>>.
- SOUZA, A. N. d. P. e. *Um framework para o cálculo de métricas de caracterização de dados de filtragem colaborativa*. 60 p. Monografia — Universidade Federal de Ouro Preto, 2014. Monografia.
- VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Editora: Independente, 2020.
- VERCEL. Next.js documentation. *Next.js Documentation*, 2021. Disponível em: <<https://nextjs.org/>>.