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

LUCAS NATALI MAGALHÃES SILVA
Orientador: Prof. Dr. Reinaldo Silva Fortes

RECSYSEXP: UM FRAMEWORK DE ALTO NÍVEL DE ABSTRAÇÃO PARA IMPLEMENTAÇÃO E VALIDAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

Ouro Preto, MG
2023

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

LUCAS NATALI MAGALHÃES SILVA

RECSYSEXP: UM FRAMEWORK DE ALTO NÍVEL DE ABSTRAÇÃO PARA IMPLEMENTAÇÃO E VALIDAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

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
2023

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S586r Silva, Lucas Natali Magalhaes.
RecSysExp Um Framework de Alto Nível de Abstração para Implementação e Validação de Sistemas de Recomendação. [manuscrito] / Lucas Natali Magalhaes Silva. - 2023.
82 f.: il.: color., gráf., tab.. + Códigos.

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

1. Sistemas de Recomendação. 2. Experimentação. 3. Framework. I. Fortes, Reinaldo. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004

Bibliotecário(a) Responsável: Luciana De Oliveira - SIAPE: 1.937.800

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Lucas Natali Magalhães Silva

RecSysExp: Um framework de alto nível de abstração para implementação e validação de sistemas de recomendação

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 24 de Março de 2023.

Membros da banca

Reinaldo Silva Fortes (Orientador) - Doutor - Universidade Federal de Ouro Preto
Daniel Xavier de Sousa (Examinador) - Doutor - Instituto Federal de Goiás
Anderson Almeida Ferreira (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 24/03/2023.

Documento assinado eletronicamente por **Reinaldo Silva Fortes**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 28/03/2023, às 10:09, 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 **0493049** e o código CRC **F543E401**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Dedico este trabalho aos meus pais Sérgio Luiz e Cláudia Maria por todo apoio e incentivo, sem vocês nada disso seria possível.

Agradecimentos

A conclusão deste trabalho contou com o suporte e colaboração de várias pessoas. Primeiramente agradeço aos meus pais, Cláudia e Sérgio, que desde sempre me incentivaram da melhor maneira a me dedicar nos estudos e sempre estiveram me acompanhando de perto, dando suporte em todos os momentos, sem vocês nada disso seria possível. Um agradecimento especial também para minha madrinha Dayse e aos meus tios, Graça e Márcio, que desde o início da minha graduação me ajudaram em diversos momentos. Agradeço também ao professor e orientador Reinaldo Silva Fortes pela oportunidade, orientações e aprendizados proporcionados nessa caminhada. Não posso esquecer também de todos os amigos que estiveram presentes nessa caminhada, a todos da República Complexo e BBCs100GR, um muito obrigado.

Um sistema de recomendação híbrido visa combinar diferentes técnicas com objetivo de obter alguma sinergia entre elas (BURKE, 2007) .

Resumo

Este trabalho consiste na implementação de um *framework* para realização de experimentos em sistemas de recomendação, seu intuito é permitir que cenários de experimentação em sistemas de recomendação possam ser criados e analisados de forma prática, isso será possível dado que o projeto fornece uma abordagem ponta-a-ponta que conta com etapas como a de pré-processamento dos dados de entrada, modelagem e treinamento de algoritmos de recomendação, avaliação dos resultados através de diferentes métricas até a visualização dos resultados. Nesse *framework* temos um conjunto de conceitos e técnicas que serão base para criação de quase todo o projeto, como principais exemplos, destacam-se a recomendação e a reprodutibilidade de experimentos. A recomendação pode ser vista como um sistema capaz de sugerir a um usuário objetos úteis e/ou interessantes considerando um grande conjunto de opções. No caso da reprodutibilidade estamos nos referindo à capacidade de diferentes investigadores tirarem as mesmas conclusões a partir de um experimento, essa característica será garantida através do arquivo de configuração criado para o **RecSysExp**. Essa construção é baseada principalmente na extração das melhores características dentre as bibliotecas e *frameworks* que possuem componentes relacionados às variadas etapas do ciclo de desenvolvimento e experimentação em sistemas de recomendação, sendo assim, esse trabalho visa facilitar a inclusão de novos paradigmas, recomendadores, *meta-features*, métricas e outros recursos. Até o momento, esse trabalho conta com um conjunto de algoritmos base para o processo de predição e recomendação, alguns deles são: **UserKNN**, **ItemKNN**, **Bias**, **BiasedSVD**, **ImplicitMF**, **BiasedMF**, **SlopeOne**, **PopScore** e outros. Como resultados desse trabalhos foi obtida uma revisão da literatura que nos proporcionou a definição de componentes, interfaces e classes que permitem flexibilidade e extensibilidade na inclusão de novos recursos ao *framework*, uma estrutura de recomendação que abrange diferentes estratégias além de maneiras de analisar e reaproveitar os resultados de cada etapa. Além disso, a partir da definição de alguns experimentos foram encontradas diferentes formas de visualização dos resultados, cálculo e armazenamento dos resultados de predições e recomendações, esses resultados foram submetidos a um conjunto de métricas como **RMSE**, **NDCG** e **MAE**. Desses resultados, conclui-se que a base geral para o *framework* foi consolidada através de diferentes estruturas de classe que permitem a extensão do projeto, métodos base relacionados a pré-processamento, recomendação e avaliação, armazenamento dos resultados de forma padronizada garantindo que todos os artefatos gerados pelo experimentos estejam organizados e disponíveis, integração entre os projetos relacionados ao **RecSysExp**, além da documentação do *framework* e dos trabalhos relacionados.

Palavras-chave: Sistemas de Recomendação, Experimentação, Framework

Abstract

This work consists of the implementation of a *framework* for carrying out experiments in recommendation systems, its purpose is to allow that experimentation scenarios in recommendation systems can be created and analyzed in a practical way, this will be possible given that the project provides a point-of-view approach. a-tip that includes steps such as pre-processing of input data, modeling and training of recommendation algorithms, evaluation of results through different metrics to visualization of results. In this *framework* we have a set of concepts and techniques that will be the basis for the creation of almost the entire project, as the main ones we have the recommendation and reproducibility of experiments. The recommendation can be seen as a system capable of suggesting useful and/or interesting objects to a user considering a large set of options. In the case of reproducibility we are referring to the ability of different investigators to draw the same conclusions from an experiment, this characteristic will be guaranteed through the configuration file created for **RecSysExp**. This construction is mainly based on extracting the best features among the libraries and frameworks that have components related to the various stages of the development and experimentation cycle in recommender systems, therefore, this work aims to facilitate the inclusion of new paradigms, recommenders, meta- features, metrics and other resources. So far, this work has a set of base algorithms for the prediction and recommendation process, some of them are: **UserKNN**, **ItemKNN**, **Bias**, **BiasedSVD**, **ImplicitMF**, **BiasedMF**, **SlopeOne**, **PopScore** and others. As a result of this work, we obtained a literature review that provided us with the definition of components, interfaces and classes that allow flexibility and extensibility in the inclusion of new features to the framework, a recommendation structure that covers different strategies as well as ways to analyze and reuse the results. of each step. In addition, based on the definition of some experiments, we were able to find different ways of viewing the results, calculating and storing the results of predictions and recommendations, these results were submitted to a set of metrics such as **RMSE**, **NDCG** and **MAE**. From these results, we were able to conclude that the general basis for the framework was consolidated through different class structures that allow the extension of the project, base methods related to pre-processing, recommendation and evaluation, storage of results in a standardized way, guaranteeing that all artifacts generated by the experiments are organized and available, integration between projects related to **RecSysExp** in addition to the framework and work documentation related.

Keywords: Recommender Systems, Experimentation, Framework.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABNT	Associação Brasileira de Normas Técnicas
AUC	Area Under the ROC Curve
API	Application Program Interface
CB	Content-Based Filtering
CF	Colaborative Filtering
CSV	Comma-separated values
DECOM	Departamento de Computação
ELF	Ensemble Learning Framework
FWLS	Feature-Weighted Linear Stacking
GUI	Graphical User Interface
HF	Hybrid Filtering
HR	Hybrid Recommender
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MAE	Mean Absolute Error
MAP	Mean Average Precision
MOF	Multi-objective Filtering
MPE	Mean Percentage Error
MRR	Mean Reciprocal Rank
MTJ	Matrix Toolkit Java
NDCG	Normalized Discounted Cumulative Gain
RMSE	Root Mean Squared Error
RS	Recommender System

STREAM	Stacking Recommendation Engines with Additional Meta-features
UFOP	Universidade Federal de Ouro Preto
UML	Unified Modeling Language
YAML	Ain't Markup Language

Sumário

1 Introdução

Em um contexto no qual os usuários são bombardeados de possibilidades de escolha, sejam elas para produtos, filmes, livros, dentre diversos outros exemplos, torna-se complexa a definição de preferências assim como a descoberta de novos itens por parte do usuário. Dessa forma, Xu, Li e Josong (2010) mostraram que o crescente aumento dos recursos disponíveis na Web faz com que os usuários enfrentem problemas para descobrir e escolher novas informações. Com isso, os Sistemas de Recomendação surgiram como uma forma de tornar a experiência de usuários e empresas mais precisa e impactante facilitando o processo de escolha e descoberta de novos itens. Esse processo é feito a partir de diferentes abordagens como, por exemplo, as filtragens colaborativa, baseada em conteúdo e híbrida, cada uma delas usufruindo de algum critério relevante para recomendação. Tais critérios podem ser baseados em informações de usuários, itens, preferências de usuários, dentre outras. Uma **preferência** pode ser definida como a relação de quanto um usuário gosta ou não de um determinado item considerando seu histórico (JUNG; HONG; KIM, 2005). Um **item**, por sua vez, é a representação de um objeto aplicado a seu contexto, ou seja, os itens de uma livraria são seus livros, os de um *e-commerce* são os produtos vendidos e assim em diante.

Além de diferentes abordagens, existem diferentes formas de avaliar as recomendações. Na visão de um usuário pode não importar apenas a acurácia que o modelo gera, ou seja, o quanto o usuário gosta de um determinado item. A avaliação pode ir bem além desse ponto e começar a introduzir aspectos como o de *novidade* e *diversidade*. Esses aspectos se mostram importantes no desempenho do modelo quando aliados à precisão de predição (FORTES; FREITAS; GONÇALVES, 2017). A novidade e diversidade, vão dizer respeito sobre o quanto uma recomendação pode surpreender de forma positiva ao indicar para o usuário, por exemplo, um filme ou produto que o usuário nunca ouviu falar ou mesmo indicar uma categoria diferente daquelas que o usuário tem costume. Sabendo que existem diferentes formas de fazer essa avaliação, muitas delas podendo ser complementares, é importante garantir que os sistemas de recomendação serão avaliados sobre diferentes aspectos, visto que essa avaliação sob óticas diferentes tende a ser vantajosa do ponto de vista de entendimento do perfil do usuário.

Tendo em vista as diferentes abordagens de implementação e perspectivas de avaliação que podem ser contempladas no contexto dos Sistemas de Recomendação, torna-se imprescindível o estudo e desenvolvimento de técnicas e processos que tornem possível a combinação dos melhores recursos para cada tipo de cenário. Com isso, o **RecSysExp** foi concebido para ser um *framework* que possibilitará a abstração de diferentes abordagens e técnicas relacionadas ao contexto de Sistemas de Recomendação, utilizando de recursos já existentes e também implementações próprias para garantir a reprodutibilidade de experimentos relacionados a recomendação com o máximo de praticidade e flexibilidade para o experimentador. Sendo assim, o intuito é que em

cada módulo do **RecSysExp** exista uma gama de recursos que permitam experimentar e testar os resultados de diferentes abordagens e técnicas, pois, assim, o experimentador poderá avaliar e decidir sobre as melhores configurações para seu experimento, garantindo que os melhores resultados sejam obtidos.

Neste trabalho, um aspecto bem relevante que será introduzido é a reprodutibilidade de experimentos. Garantir que essa característica exista é importante, visto que, quando algum trabalho é submetido para o público ele pode conter um conjunto de resultados específicos para os dados, algoritmos e métricas utilizadas. Alguém que vai desenvolver um estudo, seja comparativo, complementar ou que vai para outra direção na mesma linha de pesquisa, pode querer reproduzir esses resultados de forma simples a fim de fazer modificações ou comparações. Por isso é muito importante que o trabalho permita esse tipo de ação sem grandes esforços. Esse processo pode ser garantido dentro desse *framework* a partir de seu arquivo de configuração, a intenção é que caso você possua o mesmo arquivo de configuração que foi utilizado para os experimentos apresentados na hora de executá-lo você encontre os mesmos resultados.

O contexto e motivação para criação deste trabalho emergem da composição de alguns trabalhos já existentes que se relacionam a área de sistemas de recomendação e/ou reprodutibilidade de experimentos, a composição desses trabalhos permitirá que no futuro um *framework* para gestão, execução e reprodutibilidade de experimentos em sistemas de recomendação seja criado. Essa ideia é proveniente dos objetivos de pesquisa de Fortes (2022), dito isso, no primeiro desses trabalhos, Nepomuceno (2014) teve como objetivo definir as principais estratégias a serem utilizadas para melhorar o processo de predição, usando para isso diferentes abordagens como a Filtragem Colaborativa e métodos *ensemble*, além de realizar experimentos em diferentes bases de dados. Em sequência, agora abordando mais o contexto dos Sistemas de Recomendação, Souza (2014) desenvolveu o *framework* **RecMetrics** que visa calcular *meta-features* por meio de Filtragem Colaborativa e Baseada em Conteúdo como, por exemplo, *Jaccard*, *Gini Index* e *Entropy* utilizando de uma arquitetura paralela para realizar esses cálculos. Dando continuidade a esse trabalho, Bruckner (2017) evoluiu o **RecMetrics** propondo um *framework* para extração de *meta-features* para bases de dados de sistemas de recomendação dividindo a arquitetura inicial de seu predecessor em etapas distintas, sendo elas preparação, processamento e saída de dados, respectivamente. Em seu trabalho, o autor busca aumentar a facilidade de configuração e execução comparado ao **RecMetrics**, possibilitar a implementação de novas *meta-features* e prover a capacidade para calcular *meta-features* em grandes bases de dados.

Pensando em gestão e execução de experimentos com praticidade e pouco custo, Pacheco (2019) propôs um *framework* que necessita apenas que sejam informadas as tarefas a serem realizadas e suas respectivas entradas, todo restante do processo que inclui a preparação, execução e exibição dos resultados das tarefas, é feito pelo *framework*. Por fim, sabendo que as avaliações dentro do contexto de Sistemas de Recomendação devem levar em conta diversos aspectos como novidade e diversidade, além do fato de que a Filtragem Híbrida pode nos levar a melhores

resultados, Fortes, Freitas e Gonçalves (2017) avaliaram diferentes estratégias de Filtragem Híbrida utilizando uma gama de características de dados como, por exemplo, os dados de entrada extras para os modelos de recomendação. Ainda relacionado a hibridização, outro conceito que estará presente neste trabalho são os algoritmos constituintes, esses podem ser definidos como os algoritmos de recomendação que contribuem como entrada para o processo de hibridização (FORTES, 2022).

Agora falando de trabalhos mais avançados que abordam diferentes estratégicas, técnicas e conceitos, podemos listar dois trabalhos bem interessantes. No primeiro deles Fortes et al. (2018) considera importantes métricas de qualidade como novidade e diversidade, apresentando uma nova e adaptável estratégia para realizar a priorização dos objetivos focado na preferência dos usuários. Nessa estratégia também são utilizadas as *meta-features*, assunto de interesse para esse *framework* e, no segundo, Fortes (2022) desenvolveu um trabalho para aprimorar a recomendação multi-objetivo em três novas perspectivas, sendo elas: caracterização dos dados de entrada, sensibilidade ao risco e priorização dos objetivos. Em seu trabalho, possui como um de seus focos avaliar o uso das *meta-features* para melhoria dos resultados do processo de recomendação. Trabalhos como esses mencionados anteriormente fogem um pouco do escopo inicial proposto neste trabalho, no entanto, deles emerge uma necessidade futura que irá guiar a expansão desse *framework* isto porque no futuro podemos inserir os recursos e experimentos desenvolvidos neste trabalho como parte desse projeto.

De posse dos trabalhos mencionados acima e tendo em vista que o nosso objetivo futuro é a criação de um *framework* para gestão, execução e reprodução de experimentos computacionais na área de sistemas de recomendação, o trabalho aqui proposto tem como objetivo tornar possível a união desses trabalhos em um único projeto em busca desse objetivo. Cada um deles terá sua responsabilidade dentro do *framework* e o intuito é adicionar uma gama de outros recursos preservando uma estrutura genérica e extensível. O **RecSysExp** é um projeto que está sendo mantido em um repositório público do **Github** no qual várias pessoas podem começar a contribuir, neste trabalho contamos com uma prova de conceito do *framework* ao permitir que os alunos matriculados na disciplina de Sistemas de Recomendação (BCC409) do Departamento de Computação da Universidade Federal de Ouro Preto pudessem inserir os trabalhos desenvolvidos por eles ao longo da disciplina. Esse uso garantiu descobertas e direcionamentos importantes no desenvolvimento e este assunto será melhor detalhado no Capitulo 3. Todo o código desenvolvido durante a criação do **RecSysExp** pode ser encontrado em repositório do GitHub[1]. Nas Seções 1.1 e 1.2 será melhor detalhado o porque este trabalho está sendo desenvolvido e também quais são os objetivos esperados ao final do mesmo.

[1] https://github.com/lucasnatali98/RecSysExp.git

1.1 Justificativa

Em um cenário onde cada vez mais as recomendações se tornam essenciais, ter a capacidade de integrar diferentes bibliotecas e *frameworks* voltados a esse tema se torna uma arma poderosa quando queremos realizar nossa própria implementação, modelagem e avaliação de sistemas de recomendação.

Este trabalho visa contribuir para a criação um *framework* de gestão, execução e reprodutibilidade de experimentos computacionais em Sistemas de Recomendação. Sabendo disso, o intuito é contribuir na construção desses experimentos provendo um *framework* flexível, extensível e fácil de usar. Este projeto ainda poderá ser base para outros estudos no futuro, pois visa englobar diferentes técnicas, estratégias e abordagens dentro do contexto dos Sistemas de Recomendação, de forma que independente do foco do trabalho, exista a possibilidade de utilizar e/ou especializar o **RecSysExp**. Além disso, neste trabalho será feito uma análise de acessibilidade e uso de ferramentas relacionadas ao projeto, permitindo a avaliação de melhorias, entendimento mais fácil dos recursos existentes e tornando a evolução mais simples.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um *framework* capaz de atuar dentro da área de sistemas de recomendação provendo recursos que permitam 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. A ideia é que de posse desses recursos seja possível fazer a gestão e execução dos experimentos, garantindo praticidade e reprodutibilidade. Para garantir tais características o *framework* será capaz de integrar todo o ciclo existente no processo de experimentação em sistemas de recomendação, com isso, diferentes trabalhos serão incorporados a esse projeto com o objetivo realizar alguma tarefa dentro desse cenário. Dito isso, a intenção é que o foco do projeto esteja na realização de experimentos ponta-a-ponta, personalizáveis e que permitam a utilização de diferentes abordagens nesse processo. Além disso, temos como objetivos específicos nesse projeto os seguintes pontos:

1. Possibilite a inclusão de novos paradigmas relacionados aos Sistemas de Recomendação de maneira simples e prática;
2. Ofereça facilidade para realização de experimentos em sistemas de recomendação;
3. Forneça base teórica e prática para evolução de trabalho um trabalho voltado a gestão, execução e reprodução de experimentos em Sistemas de Recomendação;
4. Documente os recursos desenvolvidos no projeto bem como os trabalhos incorporados a ele.

1.3 Organização do Trabalho

O desenvolvimento desse trabalho se iniciou com uma revisão bibliográfica sobre os Sistemas de Recomendação, as *meta-features* e seu impacto nesses modelos de recomendação, a definição e explicação dos principais tipos de Filtragem (Colaborativa, Baseada em Conteúdo e Híbrida), dentre alguns trabalhos relacionados para embasar a construção do **RecSysExp**. Todas essas informações estão presentes no Capítulo 2.

Após ser apresentada a revisão bibliográfica, o Capitulo 3 é responsável por definir e explicar toda a metologia de trabalho aplicada, bem como a arquitetura do *framework* proposto e outras decisões de desenvolvimento.

No Capítulo 4 apresentamos então os dados utilizados, a metodologia aplicada para os experimentos e testes, além dos resultados encontrados para a pesquisa.

Feito isso, no Capitulo 5, são apresentadas conclusões sobre a execução e experimentos feitos sobre o trabalho, bem como o aprendizado adquirido e algumas considerações sobre o futuro dessa pesquisa.

2 Revisão Bibliográfica

Este capítulo tem como objetivo explicar conceitos e processos utilizados para a realização deste trabalho. Nas próximas seções serão apresentados temas como sistemas de recomendação, *meta-features*, o processo de hibridização, utilização de *frameworks* para otimizar cenários experimentais dentre outros conceitos importantes para o entendimento do trabalho.

Na Seção 2.1 serão apresentados conceitos importantes sobre os Sistemas de Recomendação, evidenciando estratégias fundamentais nessa área como, por exemplo, a *Filtragem Colaborativa*, *Filtragem Baseada em Conteúdo* e *Filtragem Híbrida*.

Com relação às *Meta-features*, Seção 2.2, é feita uma descrição da importância e dos benefícios da utilização das *meta-features* aplicadas nos sistemas de recomendação.

Por fim, a Seção 2.3 apresenta trabalhos relacionados ao **RecSysExp**. Portanto, encontraremos trabalhos que já fazem parte do processo de construção do *framework* que é objeto de estudo do Professor Reinaldo Silva Fortes bem como outras bibliotecas e *frameworks* que irão agregar valor na construção deste projeto, seja fornecendo recursos ou base teórica.

2.1 Sistemas de recomendação

Burke (2002) em seu trabalho aborda duas diferentes concepções de como podem ser definidos os sistemas de recomendação, primeiro ele apresenta uma concepção baseada no trabalho de Resnick e Varian (1997) que definia os sistemas de recomendação como mecanismos que a partir de recomendações de usuários submetidas como entrada eram agregadas e direcionadas para destinatários apropriados. Com o tempo, Burke (2002) tornou a definição mais geral, conceituando os Sistemas de Recomendação como produtores de recomendações individuais ou sistemas capazes de guiar o usuário para objetos úteis ou interessantes dentre um grande conjunto de opções.

Atualmente vários contextos estão sendo moldados pela utilização dos Sistemas de Recomendação. Áreas como a da música, filmes e livros cresceram a partir da coleta de informações sobre as preferências dos usuários em relação a determinados conjuntos de itens. Essas preferências podem ser adquiras principalmente a partir de duas principais maneiras: explicitamente e implicitamente. Para a primeira forma o gosto do usuário é obtido a partir de suas avaliações, sendo elas positivas, neutras e/ou negativas. Quando utilizamos a forma implícita, os dados desses usuários são coletados normalmente monitorando o seu comportamento, como músicas ouvidas, aplicativos baixados, sites visitados e livros lidos (BOBADILLA et al., 2013). Esses diferentes tipos de dados interagem com este *framework* através das representações criadas para as bases de dados, cada uma delas pode-se abstrair essas informações de forma com que em etapas de

pré-processamento ou análise, por exemplo, seja possível utilizá-las.

Dentro desses diversos contextos, os usuários têm que se habituar a uma quantidade enorme de conteúdos dos mais diversos tipos. Esse volume de informação acaba gerando uma sobrecarga e os sistemas de recomendação buscam por reduzir esse problema equilibrando fatores como precisão, novidade, dispersão e estabilidade.

Quando estamos falando de gerar uma recomendação, existem alguns critérios e escolhas que precisam ser avaliadas como, por exemplo, o tipo dos dados disponíveis, o algoritmo de filtragem, o modelo escolhido, técnicas utilizadas no processo, nível de esparsidade do banco de dados e desempenho no geral (BOBADILLA et al., 2013). Para isso, existem vários métodos que desempenham um papel fundamental no processo de recomendação, sendo eles: Filtragem Colaborativa, Filtragem Baseada em Conteúdo e Filtragem Híbrida. Apesar de existirem outras abordagens, como a Filtragem Demográfica, neste trabalho não temos o intuito de aprofundar nelas, implementaremos cenários que envolvam os três tipos de filtragem mencionados anteriormente. Como uma das premissas do trabalho é desenvolver um *framework* expansível, incluir outras abordagens como a Filtragem Demográfica poderá ser feito de forma simples, visto que esforço será especializar os recursos existentes.

Nas próximas seções aprofundaremos o estudo em algumas das diferentes formas de filtragem existentes para o processo de recomendação. Na Seção 2.1.1 será detalhado aspectos relacionados a Filtragem Colaborativa, em sequência na Seção 2.1.2 falaremos das principais características da Filtragem Baseada em Conteúdo e, por fim, falaremos da Filtragem Híbrida na Seção 2.1.3.

2.1.1 Filtragem Colaborativa

A Filtragem Colaborativa é o processo de filtrar ou avaliar itens usando as opiniões de outras pessoas (SCHAFER et al., 2007). A aplicação da Filtragem Colaborativa geralmente envolve conjuntos de dados muito grandes e seus métodos tem sido aplicados em diferentes cenários como: dados de detecção e monitoramento, detecção ambiental, dados financeiros, *e-commerces*, dentre outros.

Em lojas virtuais poderíamos pensar no seguinte exemplo: um comprador adquiriu um produto qualquer e fez uma avaliação de acordo com o seu grau de satisfação, o sistema de recomendação então poderá utilizar da similaridade entre as avaliações desse usuário e também de outros para tentar inferir outros itens que possam agradá-lo.

Quando falamos da CF (do inglês *Collaborative Filtering*) podemos listar alguns pontos positivos, por exemplo, não é necessário que o modelo tenha conhecimento do domínio da aplicação, isto porque as incorporações são aprendidas automaticamente. Além disso, outra vantagem importante é em relação ao modelo não saber de fato se um usuário está interessado em um item e ainda assim conseguir recomendá-lo baseado no interesse de usuários semelhantes

(DANGETI, 2022; GOOGLE, 2020). Por outro lado, a Filtragem Colaborativa tem dificuldade em lidar com novos itens. Esse problema é conhecido como o problema de inicialização a frio, que diz respeito ao sistema não pode fazer inferência para itens ou usuários sobre os quais não possui informação suficiente.

Apesar disso, é possível utilizar de técnicas que tentam contornar esse problema. Uma delas é a *Projection in WALS* (PAN et al., 2008), que consiste no modelo considerar algumas interações com os usuários e a partir disso computar uma incorporação para o item em questão sem precisar treinar novamente o modelo e outra possível abordagem é a utilização de heurísticas para gerar incorporações para os novos itens, ou seja, caso o modelo não tenha interações, ele pode calcular uma média das incorporações para itens da mesma categoria e dessa forma gerar uma recomendação (GOOGLE, 2020).

Por fim, podemos mencionar também questões como a necessidade de um grande volume de dados sobre os usuários para fazer recomendações com alta acurácia e a dificuldade de escalar um modelo de recomendação devido à alta demanda de poder computacional (SELIM; SAHAL; ELKORANY, 2014).

2.1.2 Filtragem Baseada em Conteúdo

No caso da Filtragem Baseada em Conteúdo, o sistema faz recomendações de acordo com as escolhas passadas que o usuário teve e também gera recomendações usando o conteúdo de objetos destinados à recomendação como, por exemplo, texto, imagem e som. A partir disso, torna-se possível estabelecer similaridade entre esses objetos, sendo assim, a recomendação de itens semelhantes é mais precisa (LANG, 1995; SALTER; ANTONOPOULOS, 2006; METEREN; SOMEREN, 2000).

Utilizando ainda do exemplo das lojas virtuais mencionado na Seção 2.1.1, ao invés do usuário/comprador receber uma recomendação baseada em sua avaliação e na de outros usuários, na filtragem baseada em conteúdo, o sistema utilizará atributos do produto, como descrição, preço, vendedor, dentre outros, para tentar encontrar produtos com características similares.

A CB (do inglês *Content-Based*) se destaca por não necessitar de informações sobre outros usuários para recomendar para um usuário em questão, isto porque o intuito é que as recomendações sejam específicas, essa característica facilita tornar o modelo escalável para muitos usuários (GOOGLE, 2018). Outro ponto positivo é que esses modelos têm a capacidade de capturar interesses específicos e, com isso, podem recomendar itens que poucos usuários estão interessados (GOOGLE, 2018). Por outro lado, as recomendações são feitas considerando os interesses existentes do usuário, isso ocasiona um problema relacionado a dificuldade em expandir os interesses do mesmo (GOOGLE, 2018).

2.1.3 Filtragem Híbrida

Um sistema de recomendação híbrido visa combinar diferentes técnicas com objetivo de obter alguma sinergia entre elas (BURKE, 2007). Usando do exemplo descrito na monografia de Bruckner (2017) para explicar o que se espera como sinergia, podemos considerar a introdução de novos itens dentro da base de dados de um RS (do inglês *Recommender System*), estes não seriam recomendados por Filtragem Colaborativa, dado que não existiriam avaliações sobre eles, sendo assim, o sistema de recomendação híbrido após testar a última abordagem buscaria utilizar das características da Filtragem Baseada em Conteúdo para tentar introduzir esses itens nas recomendações de alguns usuários.

Os sistemas de recomendação híbridos podem ser divididos com base em três *design's* básicos: *Monolithic*, *Parallelized* e *Pipelined*. A partir deles podemos definir classes como, *Weighted*, *Switching*, *Mixed*, *Feature Combination*, *Feature Augmentation*, *Cascade* e *Meta-level* (FORTES, 2022). Neste trabalho o foco está no design *Parallelized* com ênfase na classe *Weighted*. Abaixo segue uma descrição sobre estas três classes do design *Parallelized*.

Weighted

Este tipo de sistema de recomendação híbrido é uma boa escolha quando os componentes de recomendação possuem poder e/ou precisão relativa consistente em todo domínio do produto, isto porque cada componente pontua um item e essas pontuações são combinadas usando um cálculo linear. A combinação feita é baseada em evidências calculadas de forma estática (BURKE, 2007).

Essa abordagem pode ser vista através de fases, na primeira acontece o treinamento de um ou mais recomendadores utilizando de um conjunto de dados de treino, logo em sequência na fase de geração de candidatos um perfil de usuário pode ser submetido a cada um dos recomendadores para que ao final sejam geradas listas de itens candidatos. De posse dessas listas podemos optar por aplicar a união ou interseção entre esses itens, esse resultado é submetido a fase de *scoring* na qual cada recomendador irá gerar um score para aquele item. Por fim, tendo os *scores* para os itens podemos aplicar pesos a eles de forma a gerar um novo *score* a partir dessa combinação.

Switching

Aqui o objetivo é selecionar um único recomendador dentre todos aqueles que constituem o processo de recomendação. A abordagem *Switching* avalia que cada componente pode ter um determinado desempenho para cada tipo de usuário. Sabendo disso, é importante que o problema seja abordado tendo definido um critério de comutação. Ele vai permitir que a troca entre recomendadores seja possível.

Mixed

Um recomendador híbrido misto reúne recomendações de seus diferentes componentes

lado a lado em uma lista combinada. Através deles é possível usar classificações reais para verificar se itens estão sendo bem classificados (BURKE, 2007). Os principais desafios neste tipo de recomendação são de apresentação das recomendações, isso porque quando estamos falando das listas de recomendações surgem questões sobre como podemos integrar esse *rankings*, sendo necessários buscar técnicas como combinar baseado na confiança do recomendador.

Neste trabalho direcionamos a atenção a permitir que os sistemas de recomendação possam trabalhar em sinergia, que todos os tipos de filtragem possam atuar em um problema de forma que ao final de possíveis experimentos seja possível visualizar o comparativo de cada uma das abordagens.

2.2 Meta-features

Apesar de existirem diferentes abordagens voltadas para recomendações de alto desempenho, encontrar maneiras de melhorar esses resultados é uma etapa crucial na construção do modelo. Dessa forma, em uma busca por resultados melhores e mais efetivos, alguns autores buscaram informações adicionais (*features*) para os modelos com o objetivo de encontrar melhores resultados para os sistemas de recomendação. Apesar dessa busca, poucos desses RS fazem uso de *meta-features* e, com isso, acabam explorando um número limitado de possibilidades referentes a abordagens e problemas (FORTES; FREITAS; GONÇALVES, 2017).

Adomavicius e Zhang (2012) concluíram que as *meta-features* de classificação podem impactar positivamente a precisão das abordagens de Filtragem Colaborativa, além disso, em seu trabalho Fortes, Freitas e Gonçalves (2017) traz uma discussão detalhada sobre a evolução dos estudos envolvendo as *meta-features* além de contribuições para a área. Em seus resultados para cenários nos quais *ratings* e itens são balanceados, os sistemas de recomendação híbridos são a melhor opção, além disso, numa avaliação geral as estratégias *Stacking Recommendation Engines with Additional Meta-features* (STREAM) e *Feature-Weighted Linear Stacking* (FWLS) obtiveram *ranks* melhores do que os métodos híbridos sem informações adicionais denominados pelos autores por HR (do inglês *Hybrid Recommender*). Sendo assim, a importância da utilização das *meta-features* é crucial visando melhores resultados.

Aliado a isso, Adomavicius e Zhang (2012) mostram que as *meta-features* impactam a CF, considerando que o melhor cenário considera espaços de classificação e densidade de dados maiores. Para as duas sub-seções a seguir serão listadas algumas das *meta-features* abordadas nos estudos e trabalhos realizados por Fortes, Freitas e Gonçalves (2017) e também por Souza (2014) e Bruckner (2017) em suas monografias. Na Seção 2.2.1 será apresentado sobre as *meta-features* baseadas em CF junto de alguns exemplos e na Seção 2.2.2 tem uma descrição análoga agora considerando as *meta-features* baseadas em CB.

2.2.1 Meta-features para Filtragem Colaborativa

As *meta-features* podem ser calculadas baseado no Item, o que diz que o valor da métrica relativo a um item i utiliza das avaliações de todos usuários que avaliaram este item i; Por Usuário, onde se utilizam os itens que foram avaliados por cada usuário u; e por último a abordagem Item-Usuário onde para cada par item-usuário utiliza das avaliações de usuários para aquele item e também de outros itens avaliados por aquele usuário (BRUCKNER, 2017). Seguem as *meta-features* definidas:

1. Proportion of Ratings
   A métrica *Proportion of Ratings* indica a proporção de avaliações de um item i sobre o número total de usuários ou a proporção de avaliações de um usuário u sobre o número total de itens.

2. Proportion of Common Ratings
   A métrica *Proportion of Common Ratings* indica o número de itens distintos que foram avaliados por usuários que avaliaram um item i em particular, dividido pelo total de itens, ou analogamente, número de usuários que avaliaram um mesmo item específico, dividido pelo total de usuários.

3. Log of Ratings Amount
   Logaritmo do número total de avaliações feitas por um usuário ou recebido por um item.

4. Índice de Gini
   O *Índice de Gini* mede o nível de desigualdade em uma distribuição de valores ordenados de forma crescente. Por exemplo, para um determinado item, podemos dizer que o índice de Gini mede quão díspares são as avaliações ordenadas deste item (HURLEY; RICKARD, 2009). No desenvolvimento do *framework*, a *meta-feature* é aplicada sobre os usuários que avaliaram um Item, sobre os itens avaliados por um usuário, ou sobre a união desses conjuntos para avaliar as desigualdade das avaliações daquele item, usuário ou par item-usuário.

5. pq - mean
   A esparsidade é intuitivamente o acúmulo de energia em poucos elementos ou coeficientes (HURLEY; RICKARD, 2009). Segundo Pastor et al. (2015), a métrica pq-mean é uma ótima medida de esparsidade, pois satisfaz aos seis requisitos propostos por Hurley e Rickard (2009). No caso dos dados de Filtragem Colaborativa, uma base esparsa quer dizer que a distribuição desses valores é pouco uniforme, onde existe concentração de valores em um número reduzido de pares. No desenvolvimento do *framework*, a *meta-feature* é aplicada sobre os usuários que avaliaram um Item, sobre os itens avaliados por um usuário, ou sobre a união desses dois conjuntos.

6. Coeficiente de variação de Pearson
   Este coeficiente é a razão do desvio padrão de uma variável, representada por exemplo pelos valores das avaliações de um determinado item ou usuário, sobre a média dos valores dessas avaliações. A métrica pode ser usada como medida de dispersão de conjuntos de usuários e itens.

2.2.2 Meta-features para Filtragem Baseada em Conteúdo

Diferente da Filtragem Colaborativa, na Filtragem Baseada em Conteúdo precisaremos considerar as informações não estruturadas que fazem parte da natureza descritiva de cada item ou usuário. Dessa forma, as métricas escolhidas devem avaliar de alguma forma esses conteúdos seja através de medidas de similaridade entre itens, usuários ou ambos. Então, podemos elencar as seguintes métricas:

1. Entropia de Shannon
   Dentro do problema estudado por Bruckner (2017), Souza (2014), Fortes, Freitas e Gonçalves (2017) a entropia calcula a coesão do conteúdo de um documento, ou seja, o quão abrangente é o assunto do qual trata. Uma entropia baixa significa que o documento trata de poucos assuntos ou que os termos contidos nele se repetem mais. Essa *meta-feature* é a única que não produz resultados que variam entre zero e um, além de ser calculada baseada nos seus próprios termos.

2. Cosseno
   A *meta-feature* Cosseno segue o ângulo entre dois vetores que representam documentos (um item). O que implica que o quanto mais próximos os vetores estiverem entre si, maior será o cosseno. Uma particularidade interessante sobre esta *meta-features* é que para ser calculada entre dois itens é preciso conhecer as frequências dos termos desse item no outro, o que cria uma dependência de uma função de busca.

3. Jaccard
   O coeficiente de Jaccard expressa a similaridade entre documentos pelo tamanho da interseção dividido pelo tamanho da união dos termos de dois documentos. A interseção é calculada como o número de termos que aparecem em ambos, enquanto a união é o número total de termos distintos.

4. Dice
   Essa *meta-feature* divide duas vezes o tamanho da interseção de dois documentos pela soma dos tamanhos de seus conteúdos.

2.3 Trabalhos Relacionados

Nessa seção estão concentrados todos os trabalhos que possuem propósito similar ou complementar a esse trabalho, seja em aspectos teóricos ou práticos. Os estudos aqui apresentados estão inseridos no contexto da área de pesquisa de Sistemas de Recomendação e/ou são ferramentas úteis para o desenvolvimento de uma aplicação nesta área. Além do trabalho desenvolvido pelo professor Reinaldo, são apresentados aqui dois trabalhos de monografia de grande importância para esse *framework*, o projeto de Souza (2014), intitulado como "Um framework para o cálculo de métricas de caracterização de dados de filtragem colaborativa" e o trabalho de Bruckner (2017), intitulado como "Framework para extração de meta-features para bases de dados de sistemas de recomendação". Esses e outros trabalhos contém pilares importantes para implementação da ideia proposta nessa monografia e serão apresentados nas subseções a seguir.

2.3.1 CBRecommender

Bruckner (2017) em seu trabalho menciona uma implementação para realizar recomendações baseadas em conteúdo, nomeada como **CBRecommender**. Nessa implementação cada base de dados passa por um programa que faz a leitura de documentos e seus termos a partir do Apache Lucene. Estruturas do tipo HashMap mapeiam todos os campos e todos os termos de todos os documentos e funções auxiliares que calculam união e interseção de conjuntos de termos. Para cada métrica a ser calculada, o sistema itera sobre os conjuntos de modo a obter as médias da métrica entre um documento e cada um dos outros.

Por ser de natureza sequencial (*single thread*) e acumular muitas estruturas em memória, este sistema apresenta um desempenho que consideramos insuficiente e um consumo de memória que torna seu uso impraticável para o *hardware* normalmente encontrado. Além disso, o fato de haver uma implementação para cada base de dados, requer esforço de programação para cada nova aplicação do problema, o que torna o software pouco flexível (BRUCKNER, 2017). Sendo assim, esse trabalho não está inserido diretamente dentro do *framework*, devido a suas limitações, ainda assim, foi estudo e mencionado aqui por ser um ponto de partida para estudos e entendimento de outros trabalhos.

2.3.2 RecMetrics

O **RecMetrics** é um *framework* capaz de calcular *meta-features* para grandes bases de dados para Sistemas de Recomendação por meio de Filtragem Colaborativa e Baseada em Conteúdo, sendo também baseado em uma arquitetura paralela para realização desses cálculos (BRUCKNER, 2017). Esse *framework* se originou do trabalho de Souza (2014) e passou uma série de evoluções realizadas por Bruckner (2017) até chegar a sua versão final.

Nele, a partir de um arquivo de texto contendo todas as avaliações de usuários e itens, é criada uma matriz de avaliações item-usuário utilizando da estrutura DataModel do *framework*

Apache Mahout. Inicialmente é criado um vetor de preferências do arquivo texto lido, esse vetor é processado por um número arbitrário de *threads* que utilizarão os dados para calcular os resultados. Por fim, um objeto de saída é acessado concorrentemente pelas *threads* para escrever os resultados em um arquivo texto em disco.

O *framework* RecMetrics possui limitações como, por exemplo, trabalhar apenas com dados de Filtragem Colaborativa, utiliza cerca de vinte argumentos de configuração incluindo as *meta-features* a serem consideradas, arquivos de entrada e saída que serão utilizados, dentre outras informações importantes que estão descritas na Seção 3.3 quando falamos das *meta-features*. A inclusão de todos esses elementos, aliado à necessidade de intervenção no código para algumas alterações, torna-o complicado de utilizar.

Sabendo disso, Bruckner (2017), baseado na arquitetura do seu predecessor, dividiu a arquitetura do sistema em três principais etapas: preparação, processamento e saída dos dados. A ideia geral é que um conjunto de dados voltados aos sistemas de recomendação seja submetido a diferentes fases de um processo que tem como objetivo gerar um conjunto de *meta-features*. Na primeira fase será feita a preparação dos dados, essa operação consiste em realizar a leitura de configurações especificadas pelo usuário e nessas configurações está inserida o tipo de entrada de dados. Conhecer a tipagem dos dados aliado com as *meta-features* informadas pelo usuário, faz com que o sistema consiga criar as estruturas de dados necessárias para o cálculo e escrita de tais *meta-features*.

Na segunda fase é feito o processamento do recurso gerado na etapa anterior, com um número arbitrário de *threads* que tem como objetivo consumir os recursos de processamento criados, que são basicamente itens ou usuários e para cada um deles será feito o cálculo das métricas. Por fim, são gerados arquivos de texto ou binários contendo os resultados do processamento da segunda fase.

Esse trabalho concluiu que as *meta-features* de Filtragem Colaborativa não exibiram grandes diferenças de desempenho em relação ao *framework* **RecMetrics**. Porém, avaliando as *meta-features* baseadas em conteúdo, elas geraram uma grande melhoria de desempenho em relação ao *software* **CBRecommender**, principalmente pelo paralelismo empregado durante o cálculo das métricas. Independente dos resultados, a ferramenta criada conseguiu trazer avanços importantes como: facilidade de configuração e execução comparado a seu predecessor; facilidade de implementar novas *meta-features*; capacidade de calcular *meta-features* para grandes bases de dados, independe se é a partir de filtragem colaborativa ou baseada em conteúdo. Portanto, o **RecMetrics** será usado no trabalho como componente responsável pelo cálculo das *meta-features* visto que tem esse propósito e também por possuir o foco direcionado ao trabalho do Professor Reinaldo Silva Fortes.

2.3.3 Um Modelo Híbrido Adaptativo para Recomendação Aprimorada

Normalmente quando pensamos em um modelo tradicional de recomendação baseado em Filtragem Colaborativa, temos opções como a utilização de *K-Nearest Neighbor* que é responsável por buscar e analisar aqueles usuários e itens que possuem similaridade com usuários ativos. A similaridade entre padrões de avaliação entre os usuários ativos pode ser calculada usando diferentes abordagens como: *Person-Correlation*, *Cosine Similarity*, dentre outros. Porém, esses modelos podem sofrer com acurácias ruins, frequentemente associadas a dois problemas, o problema de partida a frio e dados esparsos.

Sendo assim, Selim, Sahal e Elkorany (2014) propôs um modelo para contornar esses problemas utilizando combinações lineares adaptáveis em diferentes técnicas de recomendação aliado ao uso de uma Matriz Triangular para agilizar o cálculo da similaridade. O *framework* proposto por Selim, Sahal e Elkorany (2014) se divide em algumas fases e tem suas principais definidas como:

- Gerar uma nova similaridade baseado na combinação da filtragem demográfica com a colaborativa tendo como base o usuário.
- A partir da similaridade gerada no passo anterior, serão obtidos os K vizinhos mais próximos.
- Gerar um conjunto das melhores recomendações baseado nos K melhores vizinhos mais próximos de usuários ativos.

Os resultados encontrados baseado principalmente na base de dados do MovieLens é que o modelo proposto melhora a precisão da recomendação e foi capaz de superar os problemas de esparsidade e partida a frio que enfrentam a Filtragem Colaborativa tradicional. Além disso, melhora o desempenho do processo de recomendação (tempo e espaço) usando matriz triangular para calcular a similaridade entre os usuários. Esse trabalho foi usado principalmente como base teórica para o desenvolvimento desse *framework*, pois traz em seu desenvolvimento características de filtragem colaborativa, algoritmos baseados em vizinhança e também a base de dados MovieLens.

2.3.4 Xperimentor: Um framework para o gerenciamento de execução de experimentos computacionais

Em um contexto onde precisamos gerenciar experimentos dos mais diversos tipos e com diferentes tipos de entrada, torna-se crucial termos uma boa gestão dos experimentos para que o trabalho não precise ser manual e custoso. Aliado a isso, Pacheco (2019) em seu estudo, afirmou não existir um sistema gerenciador de experimentos que possui foco na modelagem e condução de um experimento, os *framework* existentes necessitam que os experimentos estejam previamente

configurados e armazenados em máquinas virtuais, demandando uma construção manual o que gera consequências negativas como a utilização precária dos recursos computacionais. Sendo assim, Pacheco (2019) propôs em seu trabalho um *framework* para facilitar e padronizar todo processo que caracteriza a realização de experimentos, desde a construção até a análise. O intuito é que usuário do *framework* precise apenas fornecer as tarefas com suas entradas e dependências, todo o restante fica na responsabilidade do **Xperimentor**. Como esse trabalho gerencia o processo de execução de experimentos, ele será uma das partes do *framework* sendo utilizado juto a estrutura de classes definidas na Seção 3.5.

Como o **Xperimentor** gerencia o processo de execução de experimentos, ele será uma das partes desse trabalho sendo utilizado na parte de experimentação junto a estrutura de classes definidas na Seção 3.5.

2.3.5 LensKit for Python

LensKit é um projeto *open-source* que em sua primeira versão utilizava Java e posteriormente foi repaginado para trabalhar com Python (EKSTRAND, 2018). Seu objetivo principal é fornecer a estrutura necessária para processar dados, avaliar e treinar algoritmos de sistemas de recomendação. Desde sua versão Java, os autores afirmam que como principais êxitos do projeto se destacam a cobertura de testes, garantindo confiabilidade; os algoritmos modulares no qual componentes individuais podem ser substituídos e reconfigurados, dentre outros aspectos.

Além disso, durante a evolução do software alguns objetivos e critérios foram estabelecidos com o intuito de tornar o **LensKit for Python** o mais útil e utilizável possível. A construção deveria ser voltada a softwares já existentes e ferramentas padrão, ou seja, ao invés de reinventar a roda, bibliotecas como Scikit-Learn, PyTorch, TensorFlow, Pandas e PyData são usadas com intuito de prover recursos e funcionalidades úteis para o *framework*. Essa característica se assemelha muito aos objetivos desse trabalho, isso porque o LensKit contém API's (do inglês *Application Program Interface*) comuns para algoritmos de recomendação, preparação dos dados de entrada para realização de experimentos, processamento em lote, um conjunto de algoritmos de filtragem colaborativa e métricas de avaliação. Todos esses recursos são parte deste trabalho e o nosso intuito é fornecer uma gama ainda maior de recursos que contemplem a inserção de novas abordagens e estratégias.

Indo mais a fundo, o projeto tem como intuito se diferenciar de outros pacotes de recomendação que visam esconder e controlar o pipeline de dados. O **LensKit** visa possibilitar que o pipeline possa ser modificado da forma que a pesquisa demandar. Sabendo disso, os autores evidenciam que o **LensKit** possui alguns principais casos de uso:

1. Avaliação do Sistema de Recomendação
   O **LensKit** fornece a possibilidade de avaliar a eficácia dos algoritmos. Esse processo pode ser feito através de implementações de qualidade e testadas para vários filtros, interfaces de

fácil implementação que permitem comparar diferentes abordagens e técnicas, garante um integração simples com outras ferramentas bem como código para avaliação de precisão de previsão e métricas *top-N*.

2. Pesquisa do Sistema de Recomendação Off-line
   Possui características úteis para avaliação de novas técnicas de recomendação além de outros experimentos que tem o intuito de estudar o comportamento de algoritmos de recomendação.

3. Educação
   O **LensKit** foi concebido para ser um kit de ferramentas útil em contextos educacionais e os autores aplicaram em diferentes cenários, sendo eles de graduação e/ou pós-graduação.

4. Produção em pequena escala
   O **LensKit** permite o uso em serviços e protótipos de produção de pequena e média escala.

Portanto, tendo esses casos de uso e objetivos bem definidos. O **LensKit** visa fornecer métricas de avaliação, algoritmos de filtragem colaborativa, funções de processamento em lote bem como suporte a experimentos de recomendação. Devido às suas características, esse projeto será incorporado ao *framework* através de especializações que utilizam o **Lenskit** para alguma operação comum dentro do projeto como os modelos de recomendação, métricas, dentre outras funcionalidades.

2.3.6 Cornac

O **Cornac** é um projeto de código aberto Python, focado em sistemas de recomendação multimodais com objetivos duplos (SALAH; TRUONG; LAUW, 2020). O projeto tem como principais objetivos fornecer escalabilidade, reprodutibilidade, suporte multimodalidade e um conjunto de módulos que vão permitir acessar, construir, avaliar e comparar modelos de recomendação. Em relação ao conjunto de módulos principais são agrupados recursos como: validação cruzada, divisão de conjunto de dados, métricas de avaliação e recursos para organizar experimentos, utilitários de leitura, formatação, dentre outros.

Sobre o suporte multimodalidade, o **Cornac** fornece rotinas para trabalhar com processamento de textos, imagens, grafos. E caso precisemos abstrair outras modalidades, o projeto permite generalizar modalidades através de uma classe chamada *FeatureModality*. Por fim, a **Escalabilidade** é garantida pela utilização de amostragem em mini-lotes para usuários, itens e derivados. Usa do ecossistema Python para fornecer recursos rápidos e seguros para garantir o desempenho da aplicação e sobre a **Reprodutibilidade**, o **Cornac** oferece suporte a pesquisas reproduzíveis, usando de algoritmos existentes e conjuntos de dados integrados.

No caso do **Cornac**, ele não está inserido de fato dentro do *framework*, ainda assim, ele foi usado como objeto de estudo para o desenvolvimento desse trabalho. Dele, buscamos entender

como são definidas as estruturas de recomendação, pré-processamento, avaliação, utilização dos conjuntos de dados integrados e também da reprodutibilidade dos experimentos no projeto.

2.3.7 Surprise

Surprise é uma biblioteca Python para construir e analisar algoritmos de previsão de classificação (HUG, 2020). Essa biblioteca fornece a possibilidade do usuário implementar sua própria técnica de recomendação usando muito pouco código. Projetado para pesquisadores com objetivo de testar novas ideias de recomendação, **Surprise** possui API leve com primitivas simples que fornece recursos e conjuntos de dados para auxiliar no desenvolvimento de uma predição. Dentre seus principais recursos podemos listar pontos como: algoritmos baseados em similaridade, baseados em fatoração de matrizes, conta também com um ferramental que possibilita a avaliação e seleção de modelos assim como pesquisa automática de hiper parâmetros.

O **Surprise** é usado neste trabalho principalmente na especialização de classes que o utilizam de alguma forma, seja nos modelos de recomendação, validação cruzada e/ou suas bases de dados. Dessa forma, ele pode estar presente em diferentes módulos do *framework*.

2.3.8 Uma análise experimental sobre sistemas de filtragem colaborativa

O trabalho proposto por Nepomuceno (2014) teve como objetivo definir as principais estratégias a serem utilizadas para melhorar a qualidade da predição. Para atingir seu objetivo, o autor se propôs, através de experimentos em diferentes bases de dados, aplicar modelos de filtragem colaborativa além de métodos *Ensemble/Blending* que consistem em um conjunto de classificadores que combinam diferentes respostas para classificar um novo exemplo/caso.

Para realizar a sua proposta, foi utilizado o *framework* Apache Mahout para realizar as predições e o *framework Ensemble Learning Framework* (ELF) para geração do RMSE (do inglês *Root Mean Squared Error*) dos algoritmos *Ensemble*. Para o trabalho (NEPOMUCENO, 2014) foram escolhidos os seguintes algoritmos presentes no Mahout: *Singular Value Decomposition*, *Slope One*, *Generic User Based*, *Generic Item Based* e *Linear Interpolation*. Além desses algoritmos, outros *ensembles* foram objeto de estudo para esse trabalho: *Decision Trees*, *Linear Regression*, *Polynomial Regression*, *Neural Network*, *Kernel Ridge Regression* e *Bagging*. Tendo conhecimento desses métodos que foram utilizados, uma estrategia de experimentação baseada em 5 passos foi definida. Cada um dos passos é diretamente dependente do seu anterior e estão organizadas da seguinte forma:

1. Seleção de dados para treino e teste
2. Divisão dos dados de treino em *5-folds*
3. Execução dos algoritmos de filtragem colaborativa

4. Seleção dos algoritmos de Filtragem Colaborativa para geração da matriz de *features*
5. Execução do *framework* ELF para geração do RMSE dos algoritmos *ensemble*

Após realizados todos os experimentos considerando bases de dados distintas e com características bem diferentes uma das outras, foi concluído que dentre os algoritmos de Filtragem Colaborativa, o que melhor performou foi o *Slope-One*. Em relação aos algoritmos *ensemble*, na maioria das vezes eles obtêm resultados melhores de RMSE comparados a modelos de Filtragem Colaborativa, sendo assim, o melhor algoritmo *ensemble* performou 24,4 % melhor do que o algoritmo de CF.

Por fim, foram feitas outras comparações entre diferentes bases de dados a fim de entender a performance de cada tipo de abordagem como, por exemplo, para as bases de dados **Movielens** e **Yelp**, todos os algoritmos *ensemble* retornaram valores de RMSE menores que os algoritmos de Filtragem Colaborativa e para a base de dados **Jester**, as Redes Neurais não tiveram bons resultados, afetando no *bagging* que levou a uma perda de cerca de 45% no RMSE.

O trabalho Nepomuceno (2014) está inserido nesse *framework* como referência teórica, seu trabalho faz a utilização diferentes algoritmos *ensemble*, divisão da base de dados em *folds*, uso de algoritmos de filtragem colaborativa e outros recursos que são do interesse desse projeto. Com isso, no futuro poderemos replicar os experimentos feitos por Nepomuceno (2014) ou também abstrair esse *framework* para utilizar o Apache Mahout.

2.3.9 LibRec

Guo et al. (2015) propuseram o **LibRec** que é uma biblioteca Java, *open-source*, que implementa vários algoritmos de recomendação além de um conjunto de métricas de avaliação. O projeto permite o trabalho em cima de dois problemas bem conhecidos em Sistemas de Recomendação, a predição de *ratings* e recomendação de itens. Sua arquitetura está projetada de acordo com a Figura 2.1:

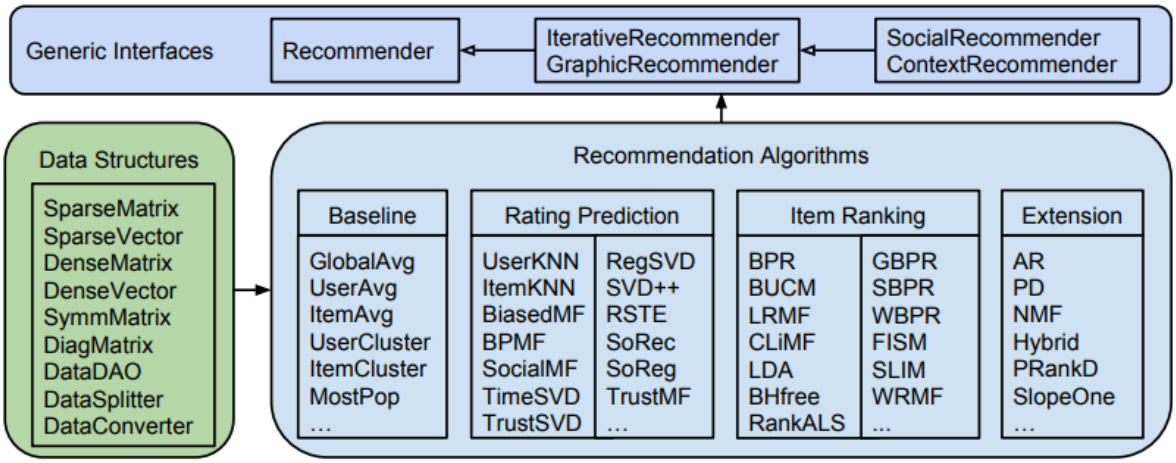

Fig. 1. The Class Structure of the LibRec Library

Figura 2.1 – Representação da estrutura de classes do LibRec

Nessa arquitetura, podemos perceber três elementos principais: **interfaces genéricas**, **estruturas de dados** e **algoritmos de recomendação**. Nas **interfaces genéricas**, são definidos conjuntos de recomendações abstratas como: *Recommender* (define um recomendador genérico estendido por *baseline* e outros algoritmos), *IterativeRecommender* (geralmente baseado em aprendizado iterativo), *GraphicRecommender* (adequado para modelos gráficos probabilísticos), *SocialRecommender* (define um recomendador que incorpora informações sociais) e *ContextRecommender* (define um recomendador que integra informações contextuais adicionais em recomendação como, por exemplo, informações temporais).

Em um contexto onde estamos trabalhando com os Sistemas de Recomendação, algumas **estruturas de dados** são amplamente difundidas e usadas, sendo elas, matrizes e vetores densos ou esparsos. Tendo em vista que o tempo de execução dos recomendadores é uma parte muito importante da implementação de um modelo, os autores escolheram por usar uma biblioteca Java para manipulação de matrizes, o MTJ (do inglês *Matrix Toolkit Java*). Além disso, foram implementadas técnicas de cache, armazenamento de dados para matrizes e vetores densos usando *arrays* bidimensionais, dentre outras melhorias. Por fim, entramos na parte dos **algoritmos de recomendação** que são divididos em três tipos principais: *baselines* que utilizam de pouca informação personalizada, os algoritmos principais que são possuem abordagens de última geração baseados em contexto e interações usuário-item;

Além dos algoritmos o **LibRec** fornece várias métricas de avaliação, sendo elas dividas em 3 principais grupos: medidas preditivas baseadas em erro, medidas baseadas em classificação e outras medidas. Dentre as medidas baseadas em erro, destacam-se o *Mean Absolute Error* (MAE), *Root Mean Squared Error* (RMSE) e *Mean Percentage Error* (MPE). Já entre as medidas baseadas em classificação, podemos listar a *Mean Average Precision* (MAP), *Normalized Discounted Cumulative Gain* (NDCG), *Mean Reciprocal Rank* (MRR), *Area Under The ROC Curve* (AUC),

Precision and Recall, dentre outras. E por último novas medidas que, por enquanto, incluem uma medida de diversidade baseada em similaridade.

Portanto, apesar do **LibRec** possuir a linguagem de programação diferente da que utilizaremos para o desenvolvimento desse trabalho, ele será utilizado apenas como base teórica para a construção de diversos elementos dentro do nosso *framework*, relacionado tanto a algoritmos quanto a métricas. Apesar de que no futuro ele possa ser inserido através de alguma especialização no *framework*.

2.3.10 Classificação e Relevância em Métricas de Novidade e Diversidade para Sistemas de Recomendação

Dentro da área de estudo dos Sistemas de Recomendação, ficar restrito somente a avaliar a acurácia do modelo pode estagnar a evolução e qualidade do mesmo, isso acontece porque além de desejar uma boa recomendação em termos de precisão, um usuário quer poder variar seu leque de opções bem como ser surpreendido com recomendações que sequer tinham passado em sua cabeça.

Dessa forma, estudos como o de Vargas e Castells (2011) começam agregar valor a literatura dessas métricas, visto que apesar do seu crescimento, a avaliação delas ainda sofre pela falta de consenso sobre suas distinção, equivalências e/ou relações. Independente disso, Vargas e Castells (2011) apresenta uma estrutura na qual foram definidas e generalizadas as métricas de novidade e diversidade, que são definidas baseado em alguns critérios como: **escolha**, **descoberta** e **relevância**.

Um ponto de deficiência dessa métricas se dá pelo fato delas perderem propriedades importantes como levar em consideração a classificação dos itens recomendados ou se os itens são relevantes ou não ao avaliar a novidade e a diversidade das recomendações. A classificação e a relevância do item são introduzidas por meio de um modelo probabilístico de navegação de recomendação, baseado nos mesmos três conceitos básicos. Com base na combinação de elementos e nas premissas do modelo de navegação, diferentes métricas e variantes se desdobram. Então, antes explorarmos de fato a representação e concepção de cada uma das possíveis métricas, é interessante que a gente esteja familiarizado com as noções do que podemos assumir como sendo novidade e diversidade.

Para Vargas e Castells (2011) novidade e diversidade são noções diferentes, embora relacionadas. A novidade de uma informação geralmente se refere ao quão diferente ela é em relação ao que foi visto anteriormente por um usuário específico ou por uma comunidade como um todo. A diversidade geralmente se aplica a um conjunto de itens e está relacionada a quão diferentes os itens são em relação uns aos outros. Isso está relacionado à novidade, pois quando um conjunto é diverso, cada item é “novidade” em relação ao resto do conjunto. Além disso, um sistema que promove novos resultados tende a gerar diversidade global ao longo do tempo na

experiência do usuário; e também aumenta a "diversidade de vendas" global da perspectiva do sistema.

Agora que foi estabelecido um entendimento sobre questões chave, vamos entender como o *framework* proposto trabalha e o que cada métrica significa e resulta. Neste *framework* o primeiro ponto a ser considerado é que uma métrica de recomendação é definida como a novidade dos itens recomendados que foram escolhidos pelo usuário. O principal elemento do *framework* são os *Item Novelty Models*, esses modelos podem definir a novidade através de várias métricas diferentes, entretanto, o foco desse trabalho foi voltado para estrategias de descoberta e distância.

Para a **novidade de item baseada em popularidade** quando temos altos valores de novidade estamos nos referindo a itens de cauda longa, que os usuários não tiveram muitas interações, caso contrário, baixos valores de novidade remetem a itens populares. No caso da **novidade de itens baseados na distância** a noção de novidade é definida através de uma função baseada na distância entre um determinado item e um contexto de experiência, caso esse contexto seja representado por um conjunto de itens, poderemos formular esse cenário como a distância esperada ou mínima entre o item e o conjunto.

O trabalho de Vargas e Castells (2011) traz uma base teórica muito importante para esse projeto, isto porque assim como foi dito no início dessa seção, restringir a avaliação dos modelos somente a acurácia pode estagnar a evolução e a qualidade do mesmo. Dessa forma, conhecer outras formas de avaliação que seguem diferentes perspectivas nos permite desenvolver uma solução que seja capaz de aceitar essas novas perspectivas de avaliação.

2.3.11 Elliot: Um Framework Abrangente e Rigoroso para Avaliação de Sistemas de Recomendação Reprodutíveis

O *framework* proposto por Anelli et al. (2021) oferece uma gama de conteúdos relacionados a todo o processo de recomendação, nele conseguimos extrair módulos responsáveis tanto pelo pré-processamento dos dados quanto para avaliação dos modelos. De posse de todos esses recursos ele busca criar experimentos de forma fácil e prática através de um arquivo de configuração simples.

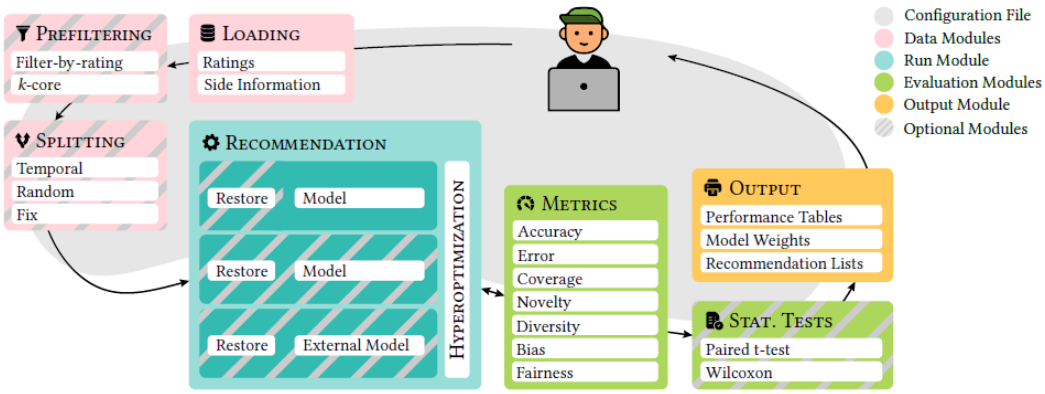

Figura 2.2 – Representação da estrutura do Elliot

Da Figura 2.2 podemos verificar que o **Elliot** oferece uma estrutura de ponta a ponta que só não engloba o processo de visualização presente neste trabalho.

Configuration 1: hello_world.yml

```
experiment:
  dataset: movielens_1m
  data_config:
    strategy: dataset
    dataset_path: ../data/movielens_1m/dataset.tsv
  splitting:
    test_splitting:
      strategy: random_subsampling
      test_ratio: 0.2
  models:
    ItemKNN:
      meta:
        hyper_opt_alg: grid
        save_recs: True
      neighbors: [50, 100]
      similarity: cosine
  evaluation:
    simple_metrics: [nDCG]
  top_k: 10
```

Figura 2.3 – Representação do arquivo de configuração

A Figura 2.3 mostra a estrutura mais simples do arquivo de configuração utilizado para realizar todo o processo dentro do *framework*, a partir dele um experimento será executado e ele pode englobar um ou mais processos.

Apesar de não estar inserido diretamente neste trabalho, o **Elliot** trouxe inspiração para o desenvolvimento desse *framework*, visto que ele entrega um resultado bem similar com o que buscamos aqui, ele parte de um ponto de partida muito similar, um arquivo de configuração que passa por etapas de pré-processamento, modelagem e avaliação de resultados, assim como foi proposto aqui. Dessa forma, no futuro, o **Elliot** pode virar alvo de uma análise comparativa entre os dois trabalhos.

2.3.12 Bibliotecas de Visualização

Quando estamos desenvolvendo algum modelo ou análise dentro de um determinado contexto, é importante que, além de dispormos de um emaranhado de dados e informações, seja possível exibir essas informações de maneira que facilite as interpretações e tomadas de decisões. Essa necessidade é muito comum dentro de cenários experimentais, nos quais é necessário validar diferentes hipóteses. Dessa forma, dispor de bibliotecas de visualização se torna uma ferramenta importante quando queremos ser mais precisos e eficientes na interpretação dos resultados. Com isso, algumas destas bibliotecas são de interesse deste trabalho.

A primeira e talvez mais importante delas é a biblioteca Matplotlib, proposta por Hunter (2007) que fornece suporte multi-plataforma, flexibilidade nos padrões de saída, alta qualidade nas visualizações, plotagens interativas, animações, classes e recursos para implementação, dentre outras funcionalidades.

Tendo como base a Matplotlib, a biblioteca Seaborn possui uma interface de alto nível para plotagem de gráficos (WASKOM, 2021). Essa biblioteca busca atribuir um alto grau de similaridade para funções que realizam tarefas diferentes, ou seja, a partir de um código hierárquico os modelos de funções atingem objetivos de visualização similares de formas diferentes. No geral, três principais módulos são definidos e basicamente todos os outros serão construídos em torno deles, são eles: *Relational*, *Distributional* e *Categorical*. Além dessas distinções, as funções também podem se agrupar a nível de figura e a nível de eixo. As funções a nível de eixo buscam substituir as funções do Matplotlib, vale lembrar que apesar de adicionarem legendas e rótulos aos eixos nenhuma outra modificação é feita neles.

Por fim, outra possibilidade muito rica de recursos é a biblioteca Plotly que cria gráficos interativos com qualidade de publicação. A biblioteca pode ser dividida em vários sub módulos. O **Plotly Express** é um sub módulo que fornece uma interface de alto nível para visualização de dados; **Graph Objects** é uma interface de baixo nível focado em figuras, traços e layouts; **Subplots** possui funções para layout de plotagens com múltiplas figuras; **Figure Factories** para construção de gráficos complexos; **I/O** é um módulo que fornece uma interface de baixo nível para exibição, leitura e escrita de figuras. Além disso, fornece um conjunto de escalas de cor, funções auxiliares, conjuntos de dados prontos para uso e construção de *dashboards* interativo que podem agrupar várias analises em um só lugar.

Conhecendo essas bibliotecas, percebemos que as que mais se encaixam em nosso cenário são: Matplotlib e Plotly. Isto porque a primeira irá fornecer principalmente os recursos relacionados a gráficos estáticos presentes neste trabalho e, aliado a isso, Matplotlib é uma biblioteca consolidada que provê uma gama de recursos que nos permitirá criar e personalizar gráficos conforme preciso. Em relação a biblioteca Plotly, seus recursos interativos, *dashboards* e também formas de visualização voltadas a área de aprendizado de máquina tornam a escolha dessa ferramenta muito importante para definição e evolução deste *framework*. Ambas bibliotecas

estarão inseridas no módulo de avaliação e visualização, fornecendo recursos para definição dos gráficos estáticos e dinâmicos, respectivamente. Essas bibliotecas foram escolhidas por abrangerem o escopo necessário para o desenvolvimento do **RecSysExp** proposta neste trabalho.

2.3.13 Discussão sobre os trabalhos relacionados

Durante toda a apresentação dos trabalhos relacionados, percebemos que a forma com que eles estão inseridos nesse projeto pode variar, eles podem estar inseridos a nível de código, inspiração ou base teórica. Trabalhos como os apresentados nas Seções 2.3.5, 2.3.4, 2.3.2, 2.3.12 estão inseridos a nível de código fonte no projeto, todos eles são usados para executar funções presentes dentro da arquitetura desse *framework*, podemos dizer então que eles tem uma participação ativa e sem eles perderíamos funcionalidades no projeto. Esse é o exemplo do **RecMetrics**, discutido na Seção 2.3.2, ele é responsável pelos resultados de todo o módulo de *meta-features*. Por outro lado, temos um conjunto de trabalhos que não estão inseridos diretamente no código fonte mas serviram de inspiração e desenho da solução, exemplos de trabalhos com esse perfil estão nas Seções 2.3.7, 2.3.11, 2.3.9. E, por fim, temos trabalhos que estão inseridos como base teórica para esse *framework*, alguns deles estão descritos nas Seções 2.3.10 e 2.3.8.

Dessa diferenciação inicial conseguimos elencar agrupamentos para os trabalhos relacionados, agora é importante abordarmos características que aproximam, diferenciam ou complementam esse *framework* nesses outros trabalhos. O primeiro trabalho que merece atenção é o **Elliot**, pois ele se aproxima muito do que está sendo proposto nesse *framework*, isto porque o trabalho utiliza de um arquivo de configuração para gerenciar a execução de experimentos que são submetidos a tarefas de pré-processamento, recomendação, testes estatísticos, dentre outras operações também presentes aqui. A principal diferença entre este trabalho e o **Elliot** é que propomos a existência de um módulo de visualização capaz de interpretar e gerar visualizações sobre os resultados gerados em diferentes etapas do *framework*, além disso, o arquivo de configuração deste *framework* contém um nível maior de informações e detalhes, tornando-o mais longo e menos direto. No entanto, a proposta é que no futuro mais uma camada de abstração seja adicionada a interpretação desse arquivo, de forma a garantir que com menos informações o *framework* ainda será capaz de criar e executar todos os processos necessários.

Outros trabalhos que merecem uma grande atenção são o **RecMetrics** e o **Xperimentor**. O primeiro assume um papel importante neste trabalho visto que será responsável por todo o cálculo das *meta-features*. Sendo assim, a estrutura desse projeto tem que estar apta a trabalhar com ele, garantindo a preparação das informações (arquivo de configuração) e execução do **RecMetrics**. No caso do **Xperimentor**, ele é um projeto que vai se comunicar com este *framework*, isto porque um *front-end* é gerado a partir da execução desse projeto e a partir dessa interface *Web* é possível construir, executar e visualizar os resultados dos experimentos definidos através de seu arquivo de configuração. O **Task Executor** é o responsável por fazer a execução das tarefas que foram definidas no arquivo de configuração inserido no *front-end*, essas tarefas são parte do **RecSysExp**

e serão executadas através de chamadas pela linha de comando. O resultado da execução de cada uma dessas tarefas, seja ela relacionado aos pré-processamentos, recomendação ou visualização será armazenado na área dos resultados do experimento, descrito na Seção 3.15.

Ainda pensando em projetos que assumem um papel complementar nesse *framework*, temos os casos de projetos como **Lenskit** e **Surprise**, descritos nas Seções 2.3.5 e 2.3.7, respectivamente. Ambos projetos são de extrema importância pois já implementam funcionalidades de interesse para esse *framework*, ou seja, através de ambos conseguimos instanciar e usar classes de recomendadores, usar métricas, validações e funções utilitárias interessantes para nosso contexto. De posse desses recursos, conseguimos especializar várias classes do *framework*, então pensando no módulo de recomendações conseguiríamos especializar classes para usar das implementações seja do **Lenskit** ou **Surprise** e posteriormente usar ambos para fazer as predições e recomendações.

O restante dos trabalhos acabam não tendo uma conexão direta com esse trabalho a nível de código, pelo menos não nesse momento, trabalhos como o **LibRec** estão implementados na linguagem Java e não tem uma conexão direta com os códigos Python, então sua execução não ocorreria diretamente no código e sim por meio da execução do projeto via linha de comando, uma situação como essa é bem possível visto que fazemos algo bem similar com o **RecMetrics** que executamos através de um arquivo jar, porém, na concepção inicial do escopo do projeto era mais interessante relacionar o *framework* com o máximo de projetos que possuíssem a mesma linguagem de programação.

Por fim, é importante frisar que esse trabalho é um projeto integrador que busca usar o máximo de recursos de outros projetos com o intuito de reaproveitar recursos, também permitir que cada vez mais recursos externos ou próprios possam ser desenvolvidos. Quando estivermos discutindo as decisões feitas no desenvolvimento do *framework* será possível entender melhor a participação de cada um desses trabalhos além de aprofundar em outras questões relacionados aos recursos existentes.

3 Desenvolvimento

Neste capitulo, serão discutidas as decisões que foram tomadas para concretizar a implementação do *framework*. Nas próximas seções será discutido sobre o funcionamento do **RecSysExp**, sua arquitetura, módulos e dependências, além de reforçar suas conexões com outros trabalhos.

Na Seção 3.1 será apresentada a visão geral do trabalho no que diz respeito à sua arquitetura, trabalhos que influenciaram esse desenvolvimento e seus casos de uso. Na Seção 3.2, estão descritas as características, tomadas de decisão e descrição dos principais módulos presentes dentro da parte de pré-processamento da arquitetura geral. Na Seção 3.3, é discutido principalmente a forma que o treinamento dos modelos pode ser feita, destacando as abordagens e o foco do *framework*. Na Seção 3.4, é explicado como é feita a avaliação utilizando-se de diferentes métricas e também como as visualizações são usadas para extrair informações sobre a base de dados e também sobre os resultados gerados. Na Seção 3.5, é descrito como se dá a organização e gestão dos experimentos, armazenamento dos resultados e o fluxo de execução.

3.1 Visão Geral

Este *framework* tem sua construção baseada nas arquiteturas e funcionalidades de outros projetos, além do material proveniente do trabalho do Professor Reinaldo Silva Fortes. Portanto, outras bibliotecas e *frameworks* possuem alta influência no desenvolvimento do **RecSysExp**, como, por exemplo, o **LensKit**, **Elliot**, **Xperimentor**, dentre outros.

Para entender melhor como se dá o funcionamento do trabalho, será apresentado sua utilização da perspectiva de atores e tarefas por meio de um caso de uso, pois, no geral, o *framework* em questão pode ser decomposto em um conjunto simples de ações que vão delimitar as possibilidades de interação com o sistema e, além da perspectiva mencionada anteriormente, é importante avaliar o projeto através de sua arquitetura geral, agora pensando realmente em etapas e módulos que depois serão concretizados em conjuntos de classes e métodos. Sabendo disso, na Seção 3.1.1, será apresentado o principal caso de uso do projeto, descrevendo um pouco sobre cada ator e as tarefas que eles podem exercer e posteriormente, na Seção 3.1.2, é descrita a arquitetura geral do projeto que visa representar de forma ampla todas as etapas que constituem o *framework* e, após apresentadas essas seções, serão discutidos aspectos mais específicos do projeto relacionado a cada uma das etapas do mesmo.

3.1.1 Casos de uso

O *framework* aqui proposto pode ser pensando a partir da utilização de dois principais atores: **desenvolvedor** e **experimentador**. Esses dois sujeitos podem ou não compartilhar de atividades dentro do *framework*. Na Figura 3.1 é apresentado um caso de uso geral do projeto que se divide em quatro principais tarefas: planejar um experimento, executar um experimento, analisar um experimento e desenvolver novas funcionalidades.

Essa divisão entre autores e tarefas visa mostrar que o *RecSysExp* busca ser de fácil utilização para o **experimentador** e também para um **desenvolvedor**. O primeiro seria um utilizador que apenas faria alterações no arquivo de configuração para preparar seu experimento e posteriormente analisar os resultados construídos pelo *framework*. Por outro lado, a figura do desenvolvedor permite a ele tanto realizar as tarefas mencionadas anteriormente quanto fazer o desenvolvimento de novos recursos. Esse fato caracteriza o desenvolvedor como alguém inserido no contexto de Sistemas de Recomendação e/ou em desenvolvimento de *software* no geral, de forma que ele consiga contribuir no código fonte do projeto, seja desenvolvendo um novo recomendador ou até mesmo corrigindo um *bug* de alguma funcionalidade existente no projeto.

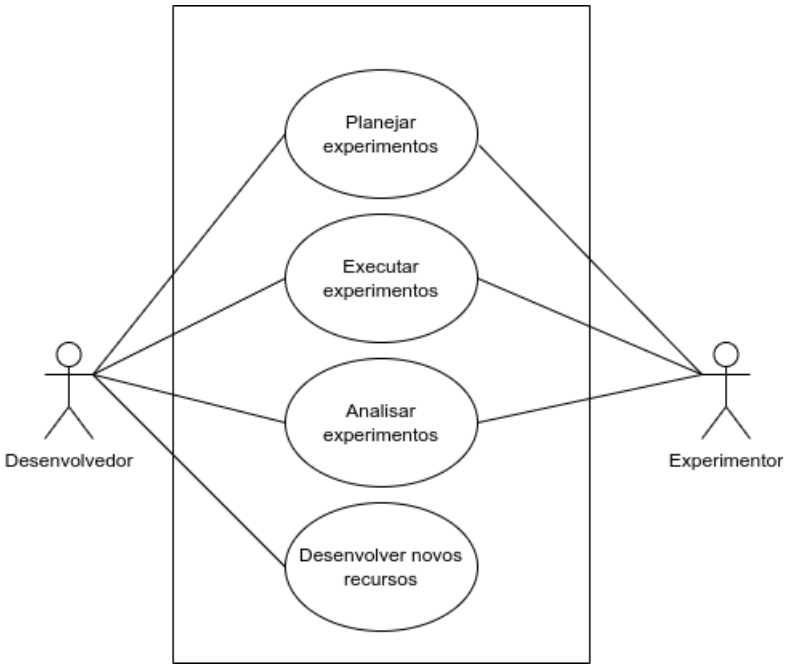

Figura 3.1 – Caso de uso geral do framework

Para entender melhor cada uma dessas tarefas vamos destrinchar o que podemos fazer em cada uma delas. No **planejamento do experimento**, o responsável precisa buscar os recursos que deseja utilizar e também avaliar quais parâmetros usar em cada situação, por exemplo, caso um dos recomendadores escolhidos seja um algoritmo baseado em vizinhança é necessário que seja especificado o tamanho máximo dessa vizinhança, outra possibilidade é que caso seja aplicada uma normalização, neste caso, é necessário pensar em qual tipo de normalização vamos aplicar. Essa etapa de planejamento realmente consiste em fazer a preparação do arquivo de configuração

no que diz respeito a informar as classes e parâmetros que estarão presentes naquele experimento.

Em relação à **execução do experimento** pode-se afirmar que ela é uma ação totalmente dependente do planejamento, considerando que esse experimento tenha como ponto de partida o arquivo de configuração. Mas, dito isso, o processo de execução considera todas as etapas que foram definidas no planejamento, então todas as instâncias de classe serão criadas e suas ações serão realizadas gerando, ao final da execução, artefatos que servirão de insumo para a interpretação dos resultados. Essa análise consiste no entendimento do que foi gerado pelo experimento. No geral, esse entendimento vai partir da visualização e entendimento de gráficos previamente definidos na etapa de **planejamento do experimento**, ainda assim, é totalmente possível que essa análise seja construída pós experimento visto que todos os resultados estarão armazenados na pasta de saída do experimento. Por fim, falando do **desenvolvimento de novas funcionalidades** essa tarefa consiste na contribuição no código fonte do projeto de forma que o colaborador seja livre para inserir novos algoritmos, pré-processamentos, corrija *bugs*, dentre outras possibilidades.

3.1.2 Arquitetura geral

Tendo conhecido o projeto da perspectiva dos casos de uso, nesta Seção, é definida a arquitetura geral para este trabalho. Ela toma como base a arquitetura proposta na tese de doutorado do Professor Reinaldo Silva Fortes. Na Figura 3.2, é apresentada a arquitetura que possui três principais subdivisões, a parte de **pré-processamento**, **modelagem e treinamento** e **avaliação e visualização dos resultados**. Cada uma delas é parte necessária para o resultado da próxima etapa.

Apresentada a arquitetura geral do projeto, nas próximas seções cada uma das etapas realizadas vão ser discutidas em mais detalhes. Sendo assim, ao final do capítulo terá sido explicado desde a fase do pré-processamento até a visualização dos resultados. A ideia é apresentar sequencialmente cada uma das etapas e ao final falar sobre como todas essas etapas podem ser utilizadas pelos experimentos. Durante essa discussão serão levantadas questões que não estão explicitas dentro da arquitetura geral, mas que ainda assim estarão presentes de alguma forma condensadas em algumas das etapas definidas.

3.2 Pré-Processamento

Na fase de **pré-processamento**, a partir de uma base de dados de sistemas de recomendação, será aplicado um conjunto de operações como, por exemplo, a normalização dos dados, seleção de dados de treino e teste, codificação de variáveis textuais, dentre outras operações. Nessa etapa de pré-processamento será definida grande parte das características que serão fornecidas para a etapa de **modelagem e treinamento** de forma que os modelos possam performar bem. A seguir, na Seção 3.2.1, é explicado como se dá o processo de carregamento dos dados junto

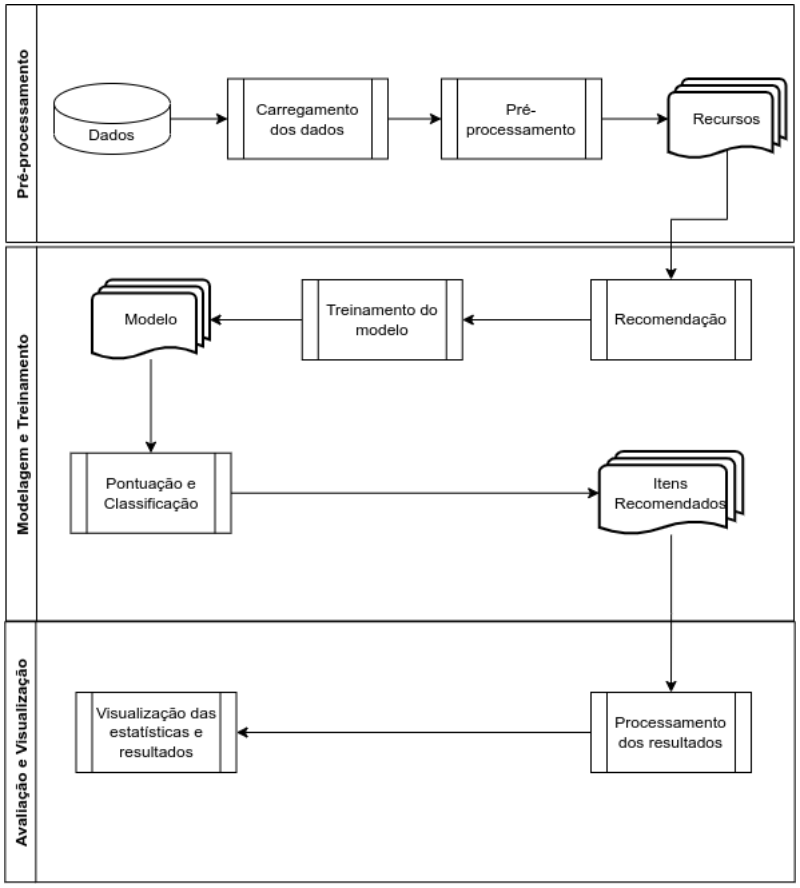

Figura 3.2 – Arquitetura geral do framework

com requisitos necessários para que outras etapas consumam esses dados e, na Seção 3.2.2, é explicado como os diferentes pré-processamentos existentes vão atuar nesses dados, gerando ao final artefatos que alimentarão etapas subsequentes.

3.2.1 Carregamento dos dados

Uma das primeiras etapas do pré-processamento é o carregamento dos dados. O *framework* proverá uma forma de carregar diferentes bases de dados e esse processo será feito a partir do módulo **datasets** em que é possível acessar diferentes classes que representam conjuntos de dados específicos como, por exemplo, o **MovieLens**. Esse processo é feito considerando uma interface chamada *Dataset* que fornece métodos comuns a serem estendidos para as diferentes base de dados.

Esses métodos foram elencados considerando que o *framework* trabalha com quaisquer dados que estejam padronizados como um **DataFrame** do Pandas, que é uma ferramenta *open source* para análise e manipulação de dados, construída em Python, utilizada em diversos contextos e trabalhos, principalmente na área de Ciência de Dados. Seu uso no projeto é justificado por possuir um conjunto de funções e recursos que facilitam a manipulação de dados, manuseio de diferentes formatos de arquivo, além de permitir várias formas de filtragem e operações nos dados. Dessa forma, através da estrutura do **DataFrame** é definida a necessidade de três *features*:

USER (contém os identificadores e atributos dos usuários), ITEM (contém os identificadores e atributos dos itens) e RATING (contém as avaliações dos usuários).

Sabendo disso, a interface *Dataset* fornece métodos que irão representar essas 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 vai possuir um conjunto de informações como idade e sexo, por exemplo. E por fim, os *items* como uma matriz que armazena para cada filme seu gênero, diretor e sinopse. Para outras bases de dados, o princípio é o mesmo e, para o caso dessas especializações precisarem de representar mais informações basta inserir novos métodos da maneira que for necessário.

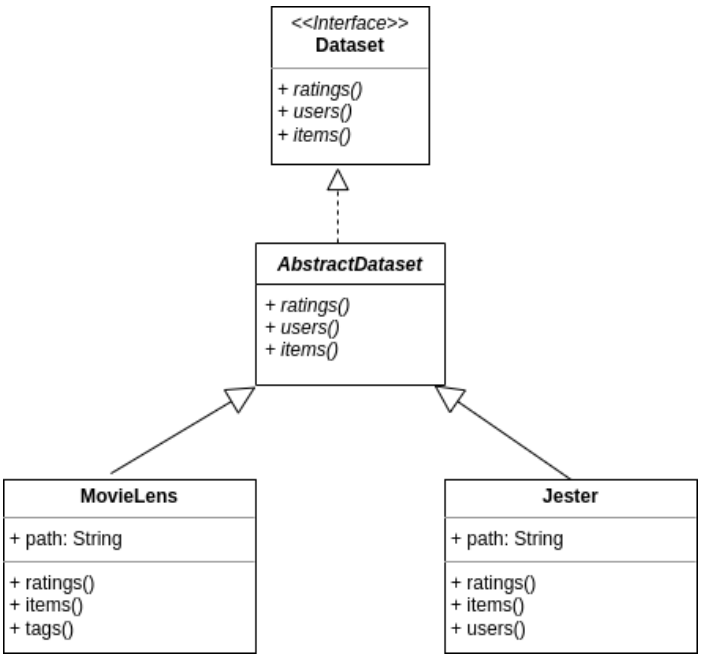

Figura 3.3 – Representação do módulo responsável pelos conjuntos de dados.

Da Figura 3.3, é possível observar que a estrutura final é baseada em uma interface genérica *Dataset* e uma classe abstrata *AbstractDataset*. Elas fornecem a base necessária para a especialização de diferentes bases de dados em forma de classes concretas. Nessas classes concretas, esses métodos listados podem trabalhar como *properties* do Python, garantindo que seja possível trabalhar com essas estruturas de ITEM, USER e RATING como atributos da classe. Por exemplo:

```
from src.data.movielens import MovieLens

movielens = MovieLens({
    'proportion': 'ml-latest-small'
})
```

```
ratings = movielens.ratings
items = movielens.items
users = movielens.users
```

Do exemplo acima, percebe-se que ao instanciar a classe do **MovieLens** definindo qual a proporção dos dados que será utilizada para essa base, é possível obter os dados referentes a itens, usuários e avaliações como se fossem atributos da classe. Quando pensamos nas possíveis especializações para as bases de dados será comum a existência de métodos específicos para o contexto de cada base, por exemplo, no caso da **MovieLens**, independente da proporção de dados existentes são utilizados arquivos *CSV* (do inglês, *Comma Separed Values*), no caso da **Jester** as bases estão definidas como arquivos *Excel*. Além de detalhes como esse, podem existir particularidades na disposição dos dados, por exemplo, que levariam cada classe a trabalhar de uma maneira específica com os dados. Então, apesar das classes compartilharem de um conjunto de métodos comuns a todas elas, pode ser que em cada classe especializada seja necessário realizar um conjunto de operações para conseguir retornar os dados da maneira que foi planejado para o *framework*.

3.2.2 Pré-processamento dos dados

Feito o carregamento dos dados, a próxima fase é a de pré-processamento, nela será possível transformar e adaptar o conjunto de dados para que seja feito o treinamento do modelo, aqui serão utilizadas diferentes estratégias para tornar os dados adequados para a fase de modelagem e treinamento. Sendo assim, na estrutura do projeto, são fornecidos arquivos que implementam classes que permitem a divisão dos dados, normalização, discretização, dentre outras estratégias.

Na Figura 3.4, é apresentada a representação da parte de pré-processamento do *framework*, assim como os demais módulos desse trabalho. Uma interface chamada *PreProcessing* fornece métodos que serão implementados pelas classes derivadas, o principal deles é o método *pre_processing*. Todos os processamentos feitos serão aplicados no atributo *dataset* da classe. Este atributo representa o conjunto de dados em questão e tem como requisito possuir as *features* que listamos quando foi discutido sobre as bases de dados: RATINGS, ITEMS e USERS.

Sabendo disso, a maneira que foi escolhida para definir os pré-processamentos foi especializar uma classe para cada tipo de processamento que será realizado no conjunto de dados, ou seja, é possível codificar, normalizar, discretizar, dentre outras possibilidades. Para cada uma delas será implementado o comportamento do método *pre_processing* de acordo com a tarefa em questão. Como existem diferentes processamentos em diferentes porções dos dados, é interessante que seja possível armazenar em uma estrutura de dados um conjunto de tarefas de pré-processamento a serem realizadas, sendo assim, a classe *PreProcessingContainer* surge para fazer a gestão desses objetos de pré-processamento que serão gerados.

Para os diferentes tipos de pré-processamento serão salvos os resultados de cada um deles na pasta responsável por armazenar a saída do experimento. Então, considerando um cenário onde será executado um experimento com dois pré-processamentos, **NormalizeProcessing** e **SplitProcessing**, ao aplicar a normalização em uma base de dados é interessante que seja armazenado o resultado da operação, de forma semelhante, se uma divisão da base de dados em treino e teste for feita é importante ter esses valores salvos. Com isso, a cada pré-processamento serão armazenados os resultados das operações temporariamente na pasta que guarda a saída do experimento, vale lembrar que também é possível obter o resultado dessas ações em memória através da chamada dos métodos. A seguir, na Figura 3.4, será apresentada a estrutura base da organização dos pré-processamentos e em sequência será apresentado um exemplo de chamada de uma dessas classes.

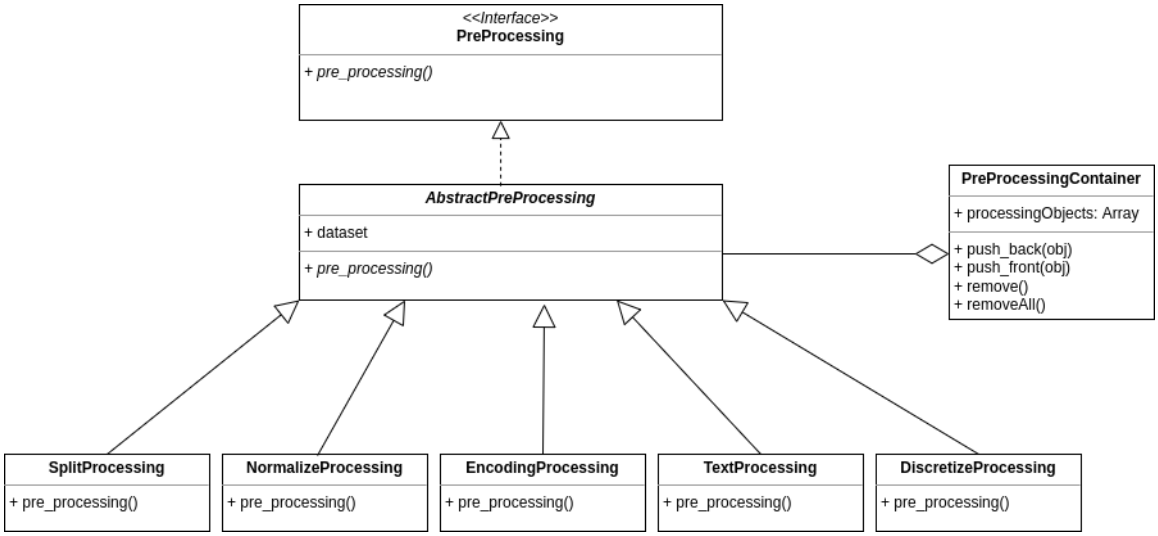

Figura 3.4 – Representação do módulo de pré-processamento

```
from src.data.movielens import MovieLens
from src.preprocessing.normalize import NormalizeProcessing

movielens = MovieLens({
    'proportion': 'ml-latest-small'
})

ratings = movielens.ratings

normalizer = NormalizeProcessing({
    'norm': 'l2',
    'column_to_apply': 'rating'
})

normalized_ratings = normalizer.pre_processing(ratings)
```

Do exemplo de código acima, uma classe de normalização está sendo instanciada e são definidos os parâmetros para a normalização diretamente pelo construtor. Nesse caso foi passado o tipo de normalização desejada e qual em *feature* será aplicado o processamento, nesse caso, os *ratings*. Passando a informação de qual *feature* será submetida a normalização não precisamos nos preocupar com essa informação na hora de chamar o método, do contrário, o método de *pre_processing* é capaz de receber parâmetros adicionais através de *kwargs* do Python (parâmetros nomeados da função) e, dessa forma, personalizar a operação da maneira que for preciso. Essa característica de poder receber parâmetros adicionais nos métodos também está inserida em outros módulos do **RecSysExp**. Na Seção 3.3, será discutido como a etapa de modelagem e treinamento irá acontecer, destacando as diferentes abordagens e também recursos que estarão presentes nessa fase do *framework*.

3.3 Modelagem e Treinamento

Na modelagem e treinamento existe uma sequência de etapas a serem seguidas para que seja possível obter os itens que serão recomendados para o usuário. Nessa fase do *framework*, estão contidos todos os algoritmos de recomendação desenvolvidos, sendo assim, todo o processo de treinamento, predição e recomendação é oriundo desse módulo. Como a proposta é generalizar ao máximo, esse módulo será responsável por implementar abordagens como a colaborativa, baseada em conteúdo, híbrida e também outras possibilidades como multi-objetivo e demográfica.

O *framework* aqui proposto permite através de um conjunto de interfaces e classes abstratas que diferentes abordagens e algoritmos de recomendação sejam implementados apenas estendendo essas interfaces e classes definidas. Abaixo, segue uma visão geral de como estão estruturadas essas interfaces e classes:

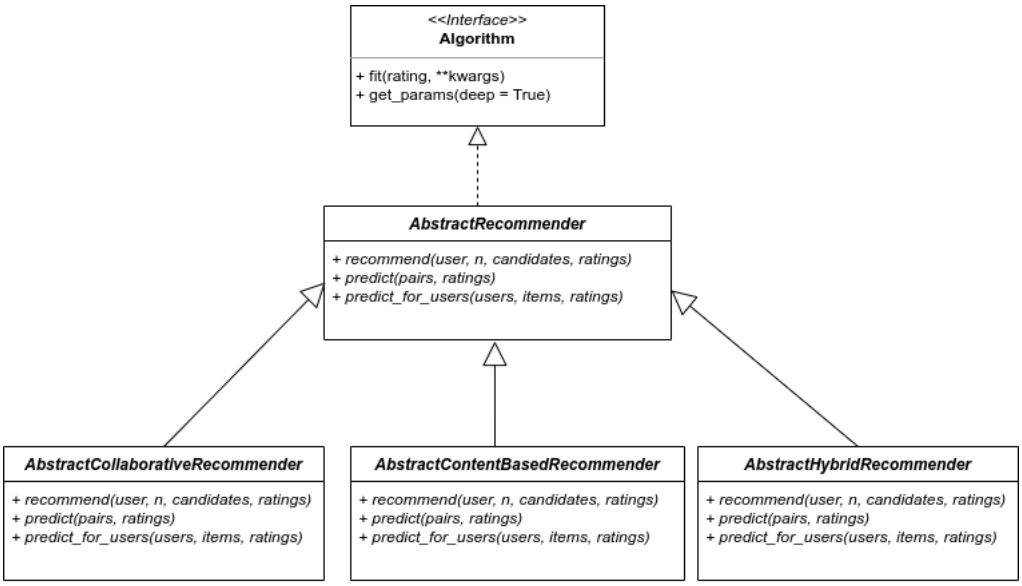

Figura 3.5 – Representação das interfaces base para recomendação

A partir da Figura 3.5, percebe-se que *AbstractRecommender* herda da interface *Algorithm*. Dessa estrutura, é possível definir as diferentes estratégias relacionadas a recomendação, ou seja, da classe abstrata *AbstractRecommender* serão derivadas as outras classes abstratas que representam cada uma dessas estratégias mencionadas: *AbstractCollaborativeRecommender*, *AbstractContentBasedRecommender* e *AbstractHybridRecommender*. Cada uma dessas classes abstratas será herdada por classes concretas que representam algoritmos dessas abordagens. Abaixo, estão relacionadas às Figuras 3.6, 3.7 e 3.8 que apresentam uma estrutura base para essas classes, vale lembrar que nem todos métodos e atributos são evidenciados para todas as classes.

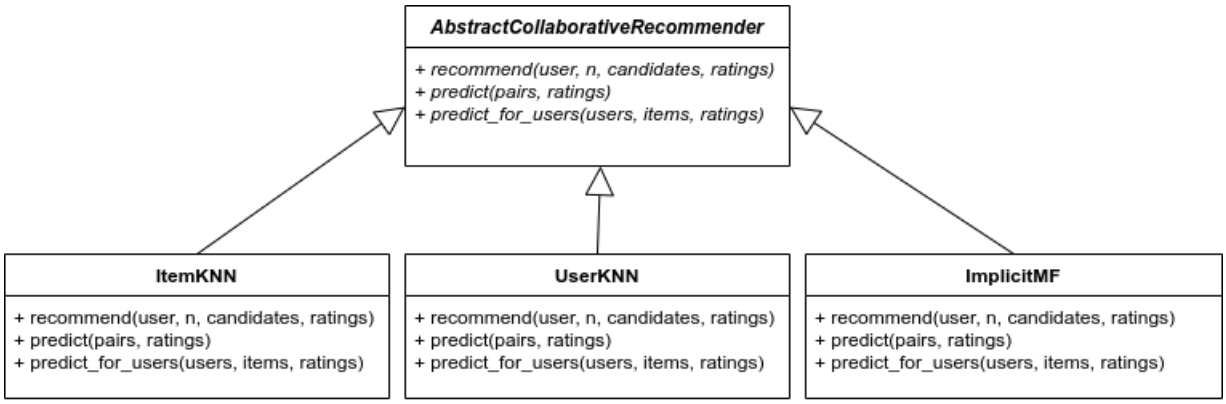

Figura 3.6 – Representação da estrutura para recomendação baseada em Filtragem Colaborativa

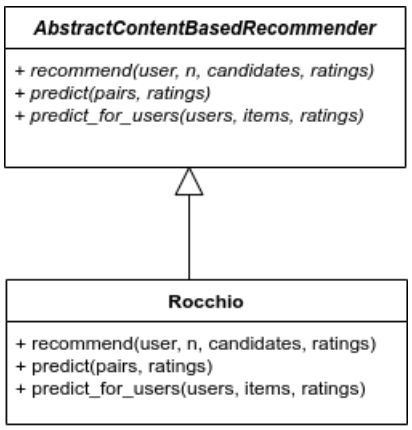

Figura 3.7 – Representação da estrutura para recomendação baseada em Filtragem Baseada em Conteúdo

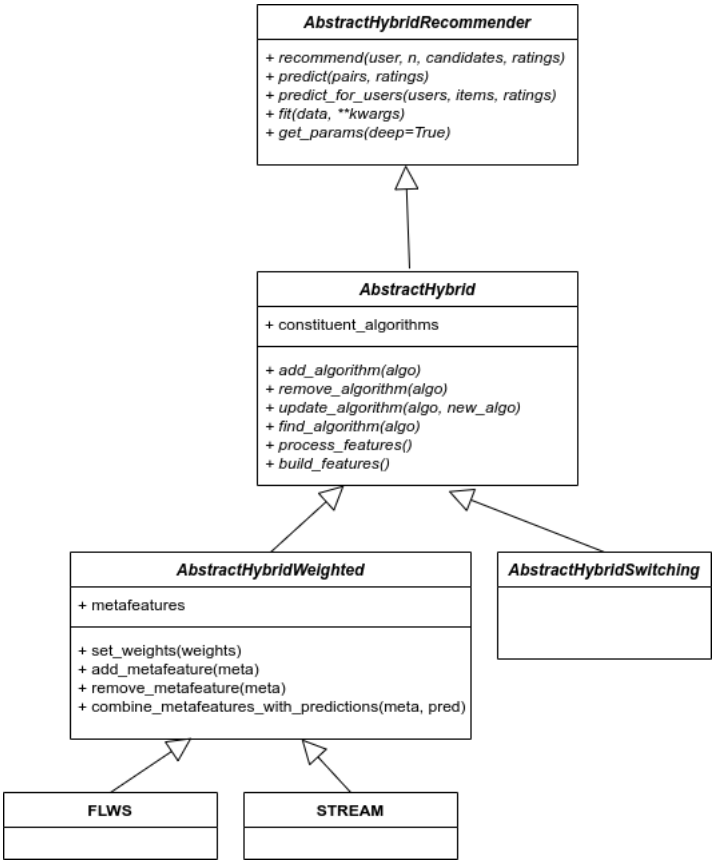

Figura 3.8 – Representação do módulo de filtragem híbrida

As imagens anteriores ilustram as classes concretas que podem ser obtidas a partir de cada uma das abordagens de recomendação, lembrando que elas são apenas uma pequena amostra do total de especializações que pode-se gerar. Essas classes serão responsáveis por gerar os resultados relacionados a recomendação, ou seja, conjuntos de predições, listas de itens recomendados para usuários, *rankeamentos*, dentre outros resultados.

Falando sobre os algoritmos de Filtragem Colaborativa, foram utilizados alguns algoritmos como: **Bias** (BIAS, 2019), **Biased-MF** (BIASEDMF, 2019), **Implicit-MF** (IMPLICITMF, 2019), **Item KNN** (ITEMKNN, 2019) e **User KNN** (USERKNN, 2019). A maioria dos algoritmos descritos anteriormente pertencem a uma classe de recomendação de algoritmos denominada **Rating Prediction**. Para a Filtragem Baseada em Conteúdo foi utilizado uma abordagem simples baseada em similaridade para obter as recomendações dos itens. No entanto, Fortes, Freitas e Gonçalves (2017) utilizaram um algoritmo que indexa o conteúdo dos itens construindo o perfil do usuário a partir do conteúdo de seu item preferido conhecido, retornando uma lista de itens classificados definidos pelas pontuações de similaridade do perfil do usuário e do conteúdo do item. Essa é uma abordagem que será inserida no *framework* em trabalhos futuros.

Pensando na abordagem de Filtragem Híbrida é considerado que diferentes *design's* a segmentam, dentro de cada design teremos um conjunto de classes e essas precisam ser implementadas para a definição do processo de hibridização. Como já dito anteriormente, este trabalho

terá como foco a classe **Weighted** da Filtragem Híbrida, pois essa abordagem é foco do trabalho de Fortes (2022). Na estrutura proposta para a representação da Hibridização, uma classe abstrata *Hybrid* implementará a interface *Recommender* e a partir de *Hybrid* podemos especializar ainda mais definindo classes como *HybridWeighted* e *HybridSwitching*, considerando a abordagem **Weighted** poderíamos ainda especializar em duas estrategias bem conhecidas, *Feature-Weighted Linear Stacking* (FWLS) e *Stacking Recommendation Engines with Additional Meta-features* (STREAM). O intuito é que no futuro outras classes como *Mixed*, *Feature Combination*, dentre outras possam ser definidas apenas considerando os métodos base já criados. Vale lembrar que a partir desse módulo podemos definir todos os algoritmos constituintes que desejarmos além de adicionar, remover e realizar outras operações com essa lista. Além dos algoritmos constituintes outra estrutura de dados será usada nesse módulo, mas agora relacionada à abordagem *Weighted*, as *meta-features*, pois nessa abordagem será possível combinar os algoritmos constituintes às *meta-features*.

Dito isso, considerando a etapa de *Cálculo das Meta-features* será utilizado o **RecMetrics** (BRUCKNER, 2017; SOUZA, 2014) para realização dessa tarefa. Sabendo que esse trabalho será usado para a realização do cálculo das *meta-features* serão discutidos aspectos mais técnicos sobre o uso e definição dos processos utilizados por ele. O primeiro detalhe que precisa ser levado em conta é que a partir do **RecMetrics** é possível extrair *meta-features* baseadas em conteúdo e filtragem colaborativa.

Apresentadas as possibilidades, serão fornecidas informações para o *framework* de forma com que todas as etapas da execução sejam executadas corretamente. Essas informações serão definidas através de um arquivo XML (do inglês *Extensible Markup Language*) que conterá um conjunto de informações para a execução, para cada processo a ser realizado é preciso que seja especificado o tipo de processamento (baseado em conteúdo ou em filtragem colaborativa) e também as *meta-features* que serão levadas em conta. Abaixo na tabela 3.1 é especificado a tabela com os valores esperados para realizar a execução do **RecMetrics**:

Tabela 3.1 – Tabela de parâmetros do arquivo de configuração do RecMetrics

Tag	Referente a	Descrição
<calculator>		Raiz do XML
<global>	calculator	Parâmetros globais de funcionamento
<process>	calculator	Define uma execução
<basePath>	global ou process	Diretório padrão do sistema
<resourceFile>	global ou process	Caminho do arquivo com dados de entrada
<bufferSize>	global ou process	Tamanho do buffer de saída (em bytes)
<outputFolder>	global ou process	Diretório dos arquivos de sa ída
<useTextOutput/>	global	Saída em formato texto (padrão binário)
<doItem/>	global ou process	Calcular *meta-features* por Item
<doUser/>	global ou process	Calcular *meta-features* por User
<doItemUser/>	global ou process (CF)	Calcular *meta-features* por Item-User
<type>	process	Tipo (collaborative ou content-based)
<metric>	process	Classe da *meta-feature* a ser calculada
<indexFolder>	process (CB)	Diretório dos dados indexados
<index>	process (CB)	Processo de Indexação (Conteúdo)
<userPreferenceFile>	process (Index CB)	Arquivo de avaliações para cálculo do tipo "User"
<userPreferenceThreshold>	process (Index CB)	Avaliação mínima para Item $\in$ User
<numThreads>	global ou process	Número de *threads* da fase de processamento

Após realizados todos os cálculos, o **RecMetrics** proverá uma saída que consiste em vários arquivos de texto separados pelo tipo de *meta-feature* com nome de cada uma delas de acordo com o que foi especificado no arquivo de configuração. Por exemplo, se for solicitado o cálculo da *meta-feature* Gini por **Item**, **Usuário** e **Item-Usuário**, será gerado um conjunto de três arquivos, sendo eles: *cf_Gini_Item.txt*, *cf_Gini_User.txt* e *cf_Gini_ItemUser.txt*. De posse desses arquivos, é possível utilizá-los no processo de hibridização, por exemplo.

Sabendo disso, a Figura 3.9 mostra o diagrama UML (do inglês *Unified Modeling Language*) do módulo de cálculo de *meta-features*. No geral, uma interface *MetaFeature* fornecerá métodos base para implementação de diferentes *meta-features*. Inicialmente serão considerados dois agrupamentos que foram mencionados anteriormente na revisão bibliográfica nas seções 2.2.1 e 2.2.2, que dizem respeito as *meta-features* baseadas em filtragem colaborativa e em conteúdo, respectivamente. Dessa forma, podemos especializar classes que representam, por exemplo, o coeficiente de Jaccard a partir da classe abstrata *AbstractContentBasedMetaFeature* ou o Índice de Gini a partir da classe abstrata *AbstractCollaborativeMetaFeature*.

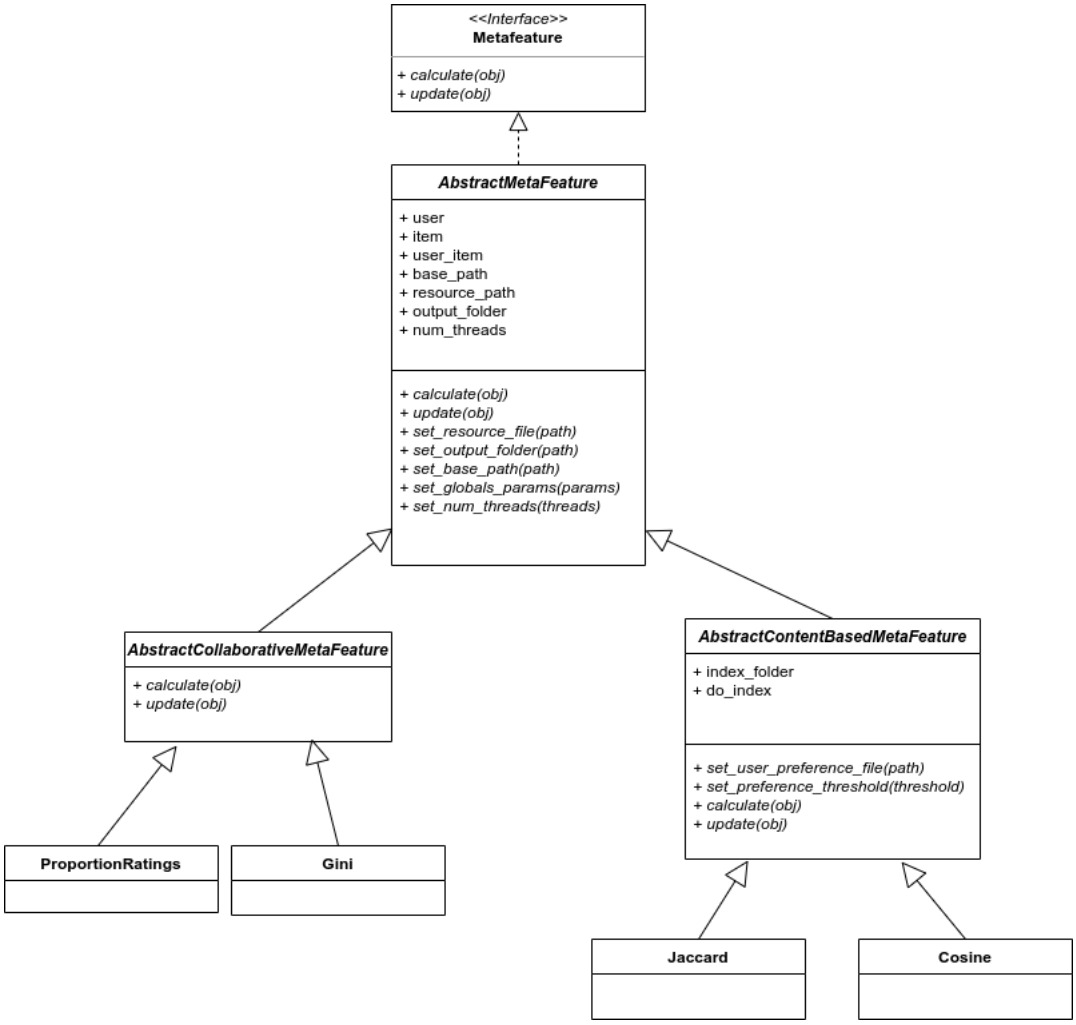

Figura 3.9 – Representação do módulo de meta-features.

Apresentadas as abordagens, podemos treinar diferentes modelos seguindo essas filtragens, os resultados desses modelos geram artefatos relacionados as recomendações. Então, para cada modelo, serão criados arquivos CSV para armazenar esses resultados dentro da pasta de saída do experimento. De posse deles, será possível nas próximas etapas avaliar cada um desses artefatos, esse assunto será tratado a seguir na Seção 3.4.

3.4 Avaliação e Visualização

Esta seção é a etapa final do fluxo do *framework*, nela serão conhecidas informações relevantes sobre o modelo definido e os itens recomendados. Em um primeiro momento é feito o processamento dos resultados com o objetivo de extrair o máximo de valor possível dos artefatos gerados utilizando de recursos estatísticos e diferentes métricas de avaliação. Em sequência será feita a visualização das informações obtidas na etapa anterior, sendo assim, na Seção 3.4.1, serão apresentados mais detalhes sobre a fase de processamento dos resultados e na Seção 3.4.2, como é possível fazer a visualização dos resultados.

3.4.1 Processamento dos resultados

Após terem sido obtidos os resultados do treinamento dos modelos de recomendação, esses resultados serão submetidos a um conjunto de métodos de avaliação. Esses métodos podem seguir os mais variados critérios para definir o que será levado em conta na hora de avaliar o resultado. Com isso, surge a necessidade de segmentar esses critérios em métricas distintas. Elas podem se dividir em grupos como **Prediction Metrics** e **Ranking Metrics**, por exemplo. Como o objetivo é permitir que sejam incluídas novas métricas, é importante definir uma estrutura na qual será fornecida uma interface *Metric* que fornece base para implementação de diversas classes de Métricas como, **RMSE**, **NDCG**, **MAE**, dentre diversas outras. Essas classes implementam as interfaces que as segmentam, ou seja, **RMSE** é uma classe que especializa a interface *PredictionMetric* e **NDCG** especializa a interface *RankingMetric*.

A Figura 3.10 mostra como será a estrutura geral dessa organização, apesar de não ter sido definida toda a amplitude da organização desse módulo, a Figura 3.10 mostra uma base de como estão definidas as classes e interfaces. Lembrando que a base para a definição do funcionamento de cada métrica pode ser definida através do método *evaluate* que especializará o comportamento para cada métrica definida.

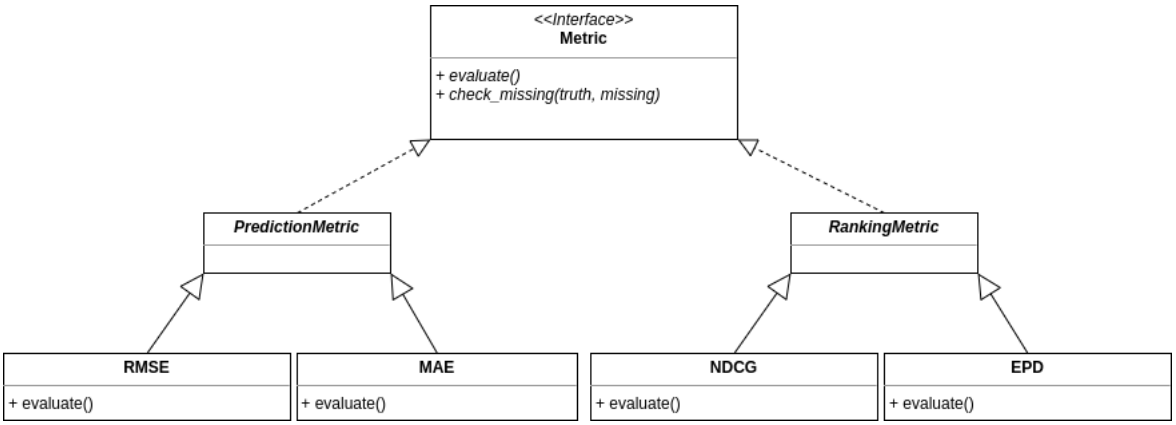

Figura 3.10 – Representação do módulo de avaliação

Conhecida a organização do conjunto de classes e interfaces que compõem as métricas, a próxima etapa do processo é submeter os resultados encontrados a partir das avaliações feitas a um módulo que possa realizar cálculos estatísticos nesses resultados. Para a realização desses cálculos sobre o que foi encontrado, existe um módulo *results* que provê todos os recursos necessários para esse processo. Dentre esses recursos, destacam-se métodos como: intervalo de confiança, anova, teste de Wilcoxon, dentre outros. Sabendo disso, a Figura 3.11 apresenta a estrutura de interface e classes definidas para trabalhar com os cálculos necessários a partir das métricas.

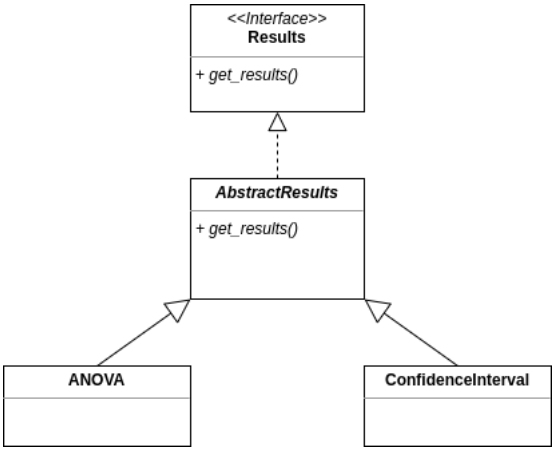

Figura 3.11 – Representação do módulo de cálculos estatísticos

De posse dos resultados obtidos da avaliação das métricas e dos cálculos estatísticos, pode-se submetê-los à próxima etapa, a visualização das estatísticas e resultados. Portanto, na Seção 3.4.2 serão apresentadas as formas de utilizar o módulo e como os resultados aqui obtidos são incorporados lá.

3.4.2 Visualização das estatísticas e resultados

Como já foi dito anteriormente, o intuito do trabalho em questão é fornecer uma abordagem *end-to-end* e permitir que, ao final dos experimentos, visualizações e estatísticas relacionadas aos modelos de Sistema de Recomendação sejam fornecidas ao usuário final como forma de mensurar os resultados obtidos. Dessa forma, para fornecer esses recursos foi necessário identificar e escolher a melhor forma de implementar visualizações que sejam personalizáveis, agrupáveis e de fácil entendimento. Com isso, na Seção 2.3.12 da revisão bibliográfica, foram discutidas características de algumas bibliotecas como: **Matplotlib**, **Seaborn** e **Plotly**.

Da análise feita sobre elas e de algumas outras bibliotecas, é comum que a maioria delas tenha sua construção baseada no **Matplotlib** e, sendo assim, ela será usada principalmente para plotagens básicas onde não precisamos de elementos interativos. Pensando em gráficos mais elaborados, a biblioteca **Plotly Python** possui um bom suporte para visualizações interativas, tendo diversas visualizações voltadas para o campo de inteligência artificial já prontas, além de permitir a criação de *dashboards* que podem trazer uma visão detalhada de algum experimento ao reunir diversas informações em um só plano.

Portanto, a ideia é que, após realizada a etapa de treinamento do(s) modelo(s), sejam gerados artefatos como os itens recomendados, que podem ser encontrados a partir da etapa de *Pontuação e Classificação* e também o resultado de técnicas de avaliação desses modelos. Com posse desses artefatos, é possível estabelecer comparações e fortalecer a tomada de decisões. Dessa forma, considera-se que, ao final da etapa de processamento dos resultados, serão encontradas diferentes análises que dizem respeito a diferentes avaliações dos itens recomendados, por

exemplo, podem ser geradas avaliações que consideram os itens novos no topo ou os itens mais diversos. E, com isso, é necessário gerar diferentes visualizações como, por exemplo, gráficos de dispersão, de barra e assim por diante.

O intuito desse trabalho é fornecer o máximo de flexibilidade e liberdade possível, com isso, o proposto para o módulo de visualizações é que ele forneça uma interface *Visualization* e uma classe abstrata *AbstractVisualization* que servem como base para definição de outras especializações. Dentre essas especializações, pode-se subdividir essas visualizações em três grupos principais: tabelas, gráficos estáticos e gráficos interativos. Desses agrupamentos surgem as classes concretas que representam formas específicas de exibir os resultados. Por exemplo, caso seja nosso objetivo trabalhar com visualizações em forma de tabelas, a classe abstrata *TablePlot* poderá ser derivada para tabelas em HTML e/ou LaTeX. A mesma ideia é aplicada para os gráficos interativos e estáticos, a partir de suas classes abstratas *InteractivePlot* e *StaticPlot*, respectivamente, é possível gerar diferentes tipos de gráficos como os de dispersão, barra, pizza, dentre outros.

Da forma que a estrutura de classes foi definida, expandir a quantidade de tipos de gráficos e tabelas até onde desejarmos é totalmente plausível. Porém, indo mais além, a inclusão de uma forma diferente de visualização como os *dashboards*, por exemplo, teria um custo baixo de inserção visto que ela poderia se comportar como uma classe abstrata que herda de *Visualization* e a partir dai implementar os métodos necessários para sua definição.

Por fim, é importante lembrar que o módulo de visualizações conta com um atributo *results*, ele contém as informações definidas após as etapas de avaliação do modelo e cálculos estatísticos. Sendo assim, as visualizações podem ser baseadas nos resultados presentes nesse atributo da classe *AbstractVisualization*. Abaixo, na Figura 3.12, temos a definição em UML da estrutura do módulo de visualização.

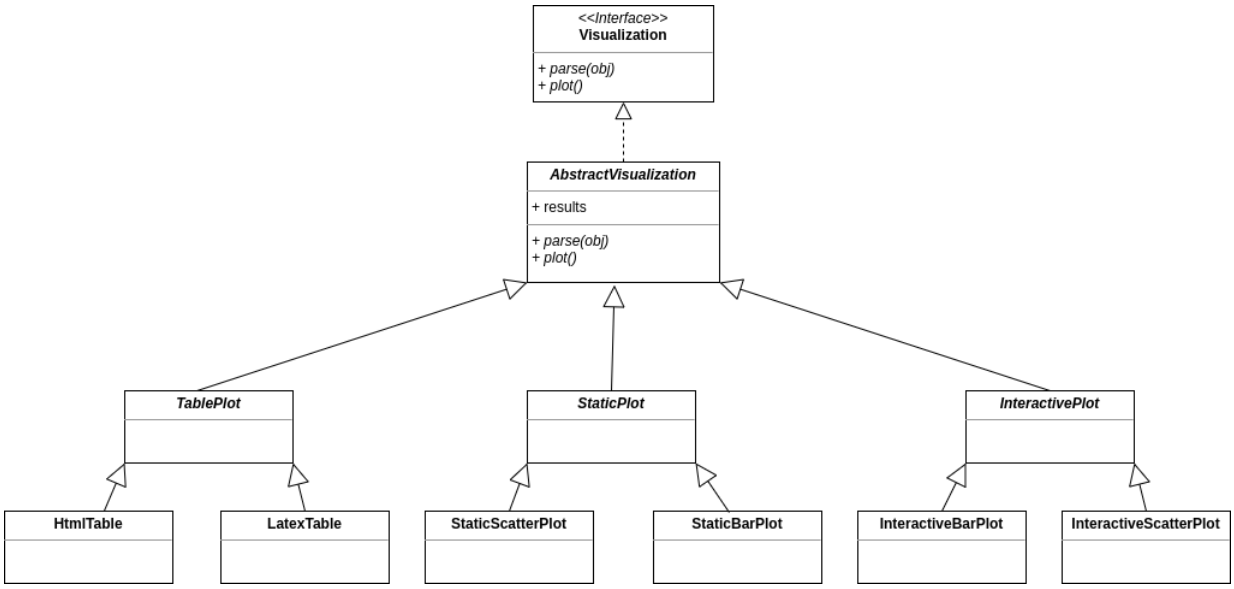

Figura 3.12 – Representação do módulo de visualização.

Agora que toda a arquitetura geral do *framework* foi apresentada, é necessário começar a falar sobre como utilizaremos todas essas funcionalidades. Um pilar que dá norte a esse trabalho é a experimentação dentro do contexto de Sistemas de Recomendação. Tendo isso em mente, será definda uma estrutura que consiga coordenar toda essa parte, esse assunto será tratado na Seção 3.5.

3.5 Funcionamento dos experimentos no framework

Da realização de cada etapa do *framework* pode-se obter diferentes informações relevantes para a realização de experimentos como, os itens recomendados, as métricas utilizas, as medidas estatísticas escolhidas e seus resultados, dentre diversas outras informações importantes. Esse montante de informações dá liberdade para que diferentes experimentos e testes sejam realizados.

Sabendo disso, o módulo que será responsável pelos experimentos do *framework* conhecerá basicamente todos os outros módulos existentes no trabalho, através dele será criada a ponte que sustenta o fato deste trabalho ser uma abordagem *end-to-end* para experimentos em Sistemas de Recomendação. Partiremos de um *Pipeline* que conterá todos os processos de cada experimento, ou seja, diferentes configurações de execução poderão ser avaliadas a partir de uma única só execução. Portanto, na Figura 3.13 é apresentada a estrutura que será responsável por esse processo.

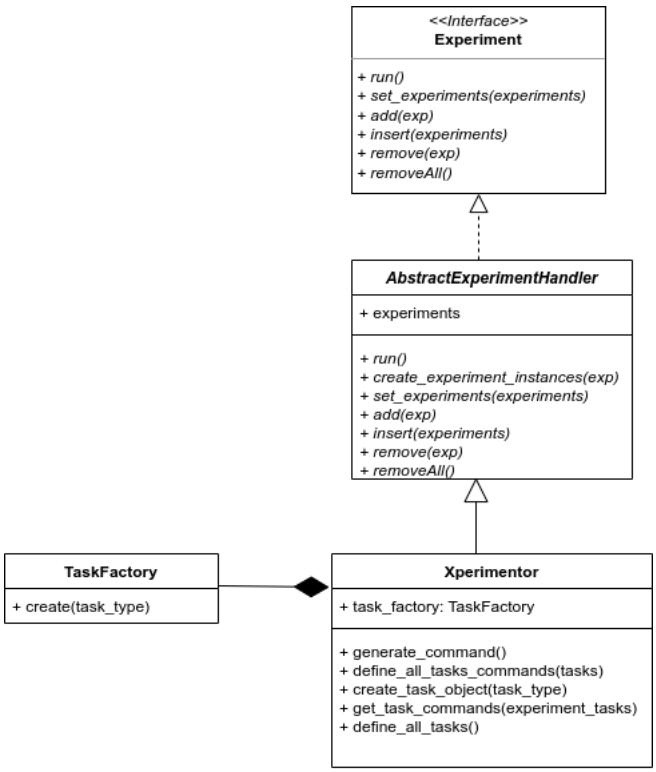

Figura 3.13 – Representação do módulo de experimentação

Da Figura acima, tem-se que *Experiment* e *AbstractExperimentHandler* servirão como

base para que todo ou qualquer projeto relacionado à gestão de experimentos possa ser acoplado ao projeto, até o momento o projeto que coordena a gestão dos experimentos é o **Xperimentor** (PACHECO, 2019). Sabendo disso, a estrutura geral dos experimentos tem como objetivo permitir que a inserção de um ou mais experimentos, a remoção deles, a criação das instâncias necessárias bem como a execução do experimento. Dentro do contexto desse trabalho existe um conjunto finito de possíveis tarefas a serem realizadas e, com isso, a classe *TaskFactory* é responsável por criar cada um dos tipos de tarefa existente. Sendo assim, a classe do *Xperimentor* pode utilizar da classe *TaskFactory* para gerar as instâncias para projetar o experimento e, para isso, conta com um conjunto de métodos que vão definir os comandos a serem executados e por ai em diante. Ainda relacionado ao funcionamento dos experimentos no *framework*, será discutido na Seção 3.5.1 sobre o fluxo de execução dos experimentos, considerando todos os elementos necessários para uma execução completa e em sequência na Seção 3.15 como os resultados de cada etapa de um experimentos são armazenados na estrutura do projeto.

3.5.1 Fluxo de execução

Neste trabalho o mais alto nível da aplicação será a parte de experimentos, nela são condensadas todas as informações que precisamos para conseguir chamar etapas subsequentes da definição dos experimentos.

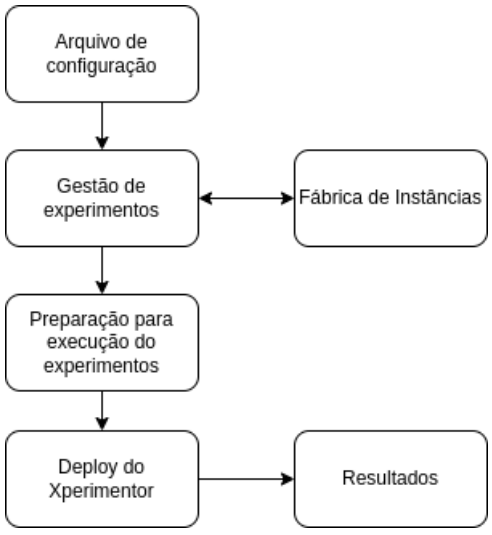

Figura 3.14 – Representação do fluxo de execução do projeto.

Da Figura 4.6 percebe-se que existe uma progressão na execução do projeto que parte do arquivo de configuração e vai até os resultados gerados. O arquivo de configuração é o ponto inicial da aplicação e quando executamos o projeto, todos os processos são baseados nas informações contidas nele.

Esse arquivo é um **JSON** que armazena as informações de todas as possíveis etapas que um experimento lida, por exemplo, é esperado que nesse arquivo sejam informados campos

que representem a base de dados, os pré-processamentos, *meta-features* a serem calculadas, recomendadores, métricas, avaliação de resultados e métodos de visualização.

Esses campos são objetos que contém, basicamente, informações sobre classes do projeto, ou seja, nome da classe, os parâmetros que ela recebe, de que módulo ela pertence, dentre outras informações. A partir dessas informações é possível instanciar objetos que, dentro do projeto, serão responsáveis pela execução de tarefas.

Vale lembrar que além das informações propriamente ditas de cada classe o arquivo de configuração contém informações do *cluster Kubernetes* onde o projeto será instanciado, a relação de dependências entre experimentos e valores **default** para serem executados em todos os experimentos também estão presentes em forma de chave nesse objeto.

Antes de entendermos mais a fundo como cada processo desse fluxo de execução funciona, vamos avaliar o exemplo abaixo que mostra um arquivo de configuração hipotético, esse arquivo está definido apenas com seu esqueleto base sem evidenciar nenhuma informação específica, isto porque a inserção de um arquivo de configuração completo tornaria a visualização no documentado prejudicada, dessa forma, um exemplo de configuração completo contendo exemplos de definição para cada chave pode ser encontrado no repositório do **Github**[1].

```
{
    "cluster_info": {},
    "experiment_dependencies": [],
    "recipesDefault": {},
    "experiments": [
        {
            "experiment_id": "exp1",
            "dataset": {},
            "preprocessing": {},
            "recommenders": {},
            "metafeatures": {},
            "metrics": {},
            "results": {},
            "visualization": {}
        }
    ]
}
```

Do exemplo acima, nota-se que esse objeto é composto por um conjunto de chaves e valores que representam informações necessárias para execução do projeto e consequentemente dos experimentos. Nesse objeto existem chaves como CLUSTER_INFO responsável por armazenar as informações para fazer o *deploy* do projeto em um *cluster Kubernetes*, RECIPESDEFAULT responsável por definir elementos que serão comuns aos experimentos, ou seja, além dos elementos que serão definidos no experimento, os valores dessa chave também serão considerados quando

[1] https://github.com/lucasnatali98/RecSysExp/blob/main/config.json

o *Xperimentor* for montar as tarefas que serão executadas. Outra chave importante desse objeto é EXPERIMENT_DEPENDENCIES que armazena as dependências entre experimentos através de uma lista de objetos e por fim a chave mais importante do objeto de configuração, EXPERIMENTS, ela é uma lista que vai guardar objetos que representam todas as etapas de um processo de experimentação.

Apresentada essa ideia inicial sobre o arquivo de configuração, serão formalizados alguns padrões utilizados. O primeiro deles é relacionado à definição de cada etapa do experimento, por exemplo, é necessário informar quais são os processos a serem executados na etapa de pré-processamento, cálculo de *metafeatures* e métricas de avaliação, por exemplo. Como todos esses processos envolvem a definição de uma ou mais classes, todos essas etapas serão armazenadas em um classe que gerencia todas as classes desse processo.

Para visualizar uma situação como essa vamos utilizar possíveis pré-processamentos que serão feitos na base de dados. A classe responsável por esse gerenciamento é *PreProcessingContainer* que possui uma lista de pré-processamentos (Representadas pela interface PreProcessing) e cada pré-processamento dessa lista possui um método *pre_processing* responsável por executar a tarefa de pré-processamento em questão.

É importante notar que existe um método padrão que todas as classes executam seguindo seu contexto. Esse padrão é compartilhado para todos os módulos desse projeto, então pode-se pensar nessa mesma lógica para a aplicação de métricas de avaliação, todas as possíveis classes executarão o método *evaluate* responsável por aplicar a métrica em questão. Todos os módulos desse trabalho com exceção das bases de dados são instanciados a partir de suas classes *container*. Isso é feito porque como podem ter vários processos dentro de uma mesma etapa, vamos armazená-las de forma agrupada para facilitar sua execução.

Na gestão dos experimentos está inserida toda a parte referente à criação de classes para representar todo o processo. Nesse momento, o módulo de experimentos usa tudo que foi criado e definido nos outros módulos para compor a execução. Então a partir da classe *AbstractExperimentHandler*, são definidos experimentos que, por sua vez, definem todos os elementos necessários para a execução do processo. Nesse momento é gerado o arquivo de configuração que é utilizado no projeto do *Xperimentor*, as instâncias das classes envolvidas no experimento são definidas, o *deploy* do projeto no *cluster* é realizado, dentre outras ações.

O processo de *deploy* é feito através de arquivos *shell* que vão subir cada um dos projetos no *cluster*, é importante lembrar que os arquivos de *deploy* já consideram que existe uma imagem já construída que pode ser utilizada para a geração dos *containers Docker*, isso quer dizer que antes de fazer a execução do projeto você tem que ter as imagens *Docker* do projeto localmente em sua máquina ou acessá-las via *Docker Hub*.

Considerando que o *deploy* tenha ocorrido de forma bem sucedida, o próximo passo da execução é fornecer o arquivo de configuração gerado pelo *framework* para o *Xperimentor*, no

momento que esse arquivo for submetido, validado, construído e executado, o *Task-Executor* vai entrar em funcionamento executando cada uma das tarefas definidas. Esses processos no geral serão arquivos *Python* que executam algum tipo de rotina ou arquivo secundário. Por exemplo, a realização dos procedimentos de pré-processamento são feitos através de uma rotina que aplica todos os processos definidos no conjunto de dados e retorna como objeto um novo conjunto de dados, por outro lado, o cálculo de *meta-features* que é feito através do **RecMetrics** que é um arquivo *jar* que receberá diversos parâmetros de linha de comando.

No momento em que todas as tarefas forem executadas pelo *Xperimentor*, pode-se considerar que o fluxo de execução do projeto terá terminado. Nesse momento, espera-se que caso todos os processos tenham ocorrido normalmente que os resultados das etapas estejam salvos na estrutura do projeto, a partir desses resultados, poderá ser feita uma analise do obtido através principalmente das visualizações definidas no planejamento do experimento.

O *framework* será capaz de executar um conjunto de tarefas pré-definidas que cobrem o escopo do projeto, então para cada etapa que temos dentro do projeto como: carregamento da base de dados, pré-processamento, aplicação dos algoritmos, cálculo das métricas, dentre outras. Existirão tarefas para representá-las, essas tarefas estão contidas em um módulo *tasks*, entre as suas tarefas temos:

1. dataset_task
2. preprocessing_task
3. algorithms_task
4. metrics_task
5. metafeatures_task
6. results_task
7. visualization_task

A tarefas mencionadas são implementadas de forma similar e seguem um mesmo princípio, toda tarefa executará um método *run* capaz de coordenar toda a execução, esse método pode conter diversas ações de preparação e manipulação dos dados, por exemplo. Sabendo disso, cada arquivo referente a uma tarefa será executado como um *script* independente através da linha de comando. Abaixo segue um exemplo de como a tarefa dos algoritmos de recomendação pode ser instanciada e executada:

```
if __name__ == "__main__":
    loader = Loader()
    config_obj = loader.load_json_file("config.json")

    experiments = config_obj['experiments']
    exp_handler = ExperimentHandler(
        experiments=experiments
    )

    experiment = exp_handler.get_experiment("exp1")
    experiment_instances = experiment.instances

    algorithms = experiment_instances['recommenders']
    algorithms_task = AlgorithmsTask(algorithms)
    algorithms_task.run()
```

Uma tarefa leva em conta um conjunto de informações que podem ter sido geradas por outras fases do *framework*, nesse caso a tarefa dos algoritmos de recomendação precisa conhecer necessariamente as bases de dados pré-processadas, para que a partir delas seja possível fazer o processo de predição e recomendação. Essa tarefa irá gerar um conjunto de artefatos que poderão ser usadas pelas etapas subsequentes. Na Seção 3.15 está detalhado como todos os resultados são armazenados em uma pasta que guarda todos os artefatos gerados durante a execução do experimento.

3.5.2 Armazenamento dos Resultados

Nessa seção é apresentado como o *framework* armazena os resultados obtidos em cada etapa de sua execução. Como existem várias tarefas a serem realizadas como: pré-processamento, cálculo de *metafeatures*, execução dos modelos de recomendação, etc. É necessário ser capaz de organizar esses resultados em diferentes pastas de forma que fique simples distinguir quais artefatos são de cada etapa. Na Figura 3.15 é apresentada a estrutura de diretórios base de um experimento.

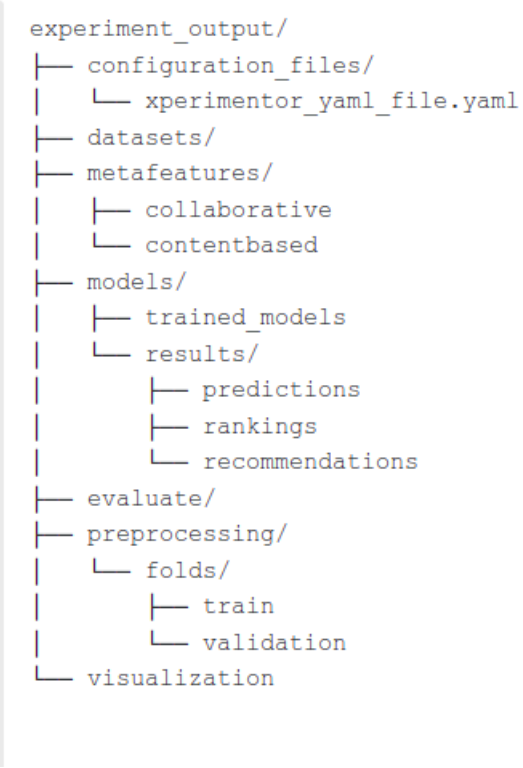

Figura 3.15 – Representação da estrutura de armazenamento dos resultados

Da imagem anterior, existem várias segmentações para armazenar os resultados, a primeira segmentação diz respeito aos arquivos de configuração gerados ao longo da execução do *framework* (*configuration_files*), os diferentes trabalhos utilizados aqui usam de formatos específicos de arquivo, sendo assim, é de responsabilidade do *framework* garantir que a partir do arquivo **JSON** seja possível fazer uma conversão correta para um arquivo **XML** e/ou **YAML** (do inglês, Ain't Markup Language), por exemplo. O próximo diretório é relacionado aos *datasets* e tem como objetivo armazenar as bases de dados após algum tipo de filtro aplicado no carregamento dos dados, por exemplo, se foi aplicado um filtro no qual devem ser selecionados apenas uma quantidade específica de registros de uma base de dados, essa nova base será armazenada nesse diretório. Em relação as *meta-features*, podemos ter uma divisão entre *meta-features* de Filtragem Colaborativa e Filtragem Baseada em Conteúdo, dito isso, os arquivos gerados para cada tipo estarão armazenadas nessas pastas.

Outro diretório muito importante é o de *models*, nele serão armazenados os algoritmos de recomendação treinados (*trained_models*), isso é feito com o intuito de reaproveitar esses modelos em algum outro momento, ainda na pasta *models* estão toda a parte de resultados que esses modelos geraram podendo ser elas as predições, *rankeamento* dos itens e também os itens recomendados. O resultado desses modelos podem ser submetidos a um processo de avaliação seja via métricas e/ou métodos estatísticos, com isso, a pasta *evaluate* vai armazenar todas as avaliações considerando a combinação dos algoritmos e métricas.

Na pasta *preprocessing* estarão armazenados todos os resultados referentes aos pre-processamentos feitos, por exemplo, caso aplicado uma estratégia de divisão da base de dados em *folds* será criado uma pasta *folds* e nela haverá tanto as bases de treino quanto de validação. Além dessa possibilidade, caso a base seja dividida usando uma estratégia *Hold-out* também vamos armazenar as bases geradas na estrutura da pasta *preprocessing*. Por fim, a pasta de visualizações é responsável por armazenar gráficos de visões que foram implementadas no projeto como, por exemplo, distribuição de *ratings* por item.

A seguir, no Capitulo 4, serão apresentados todos os resultados obtidos durante o desenvolvimento do **RecSysExp**, esses resultados estão segmentados de duas maneiras: experimentos computacionais acompanhados de uma análise sobre os recursos desenvolvidos e uma avaliação dos resultados de acordo com os estudos de caso feitos na disciplina de BCC409.

4 Resultados

Durante o desenvolvimento deste trabalho, foi construído um *framework* que tinha como objetivo definir uma estrutura genérica e extensível que fosse capaz de reproduzir experimentos computacionais envolvendo sistemas de recomendação. Desse fato, é possível analisar os resultados da ótica de construção de todo o código, documentação, recursos disponibilizados e avaliação dos resultados de alguns experimentos computacionais levando em conta diferentes modelos de recomendação, toda essa análise será feita na Seção 4.1. Além disso, como esse *framework* foi disponibilizado na disciplina de Sistemas de Recomendação, **BCC409**, do Departamento de Computação (DECOM) da Universidade Federal de Ouro Preto (UFOP), esse trabalho conta com estudos de casos e observações geradas ao longo da disciplina, esses resultados foram muito importantes para notar pontos de melhoria e também necessidades que precisam ser garantidas para a evolução e disponibilização desse *framework* como um projeto *open-source* para um público maior, esses resultados estão disponíveis na Seção 4.2.

4.1 Experimentos

Para validar o funcionamento do *framework* e também garantir que os objetivos desejados neste projeto foram atingidos, foi definido um conjunto de experimentos que abrangerão diferentes cenários para avaliar o processo de recomendação. Os experimentos serão executados em uma máquina com processador i3 e 12GB de RAM, executados localmente sem conexão com serviços de nuvem.

Para realizar esses experimentos o foco será direcionado para a abordagem de Filtragem Colaborativa por existir uma gama maior de recursos e algoritmos disponíveis, além disso, esses experimentos utilizarão diferentes proporções da base de dados **MovieLens** e para que eles fossem executados sem nenhum problema foi necessário definir amostras para essa base de dados, visto que usar grandes volumes de dados considerando bases que possuem milhões de registros seria inviável considerando o *hardware* em que o *framework* será executado para os experimentos. Dito isso, foram escolhidas as seguintes proporções: 100 mil, 200 mil, 300 mil e 400 mil registros, todas essas amostras com exceção da de 100 mil foram escolhidos a partir de uma amostragem aleatória dos dados, provenientes da MovieLens com mais de 1 milhão de registros, além disso, todas essas bases passarão por uma divisão de dados em treino e teste, seguindo a proporção de 70/30, respectivamente. Feito isso, essas bases de dados serão submetidas ao processo de predição e recomendação utilizando de diferentes algoritmos, sendo eles: **UserKNN**, **ItemKNN**, **Bias** e **BiasedSVD**, todos esses utilizando implementações do **LensKit Python**. De posse dos resultados vamos submetê-los às métricas: **RMSE**, **MAE** e **NDCG**. Além da apresentação dos resultados dos experimentos mencionados anteriormente também será feita uma discussão em

relação aos resultados obtidos do desenvolvimento em geral.

Inicialmente, serão avaliados os resultados relacionados ao desenvolvimento no geral. O **RecSysExp** tem como objetivo fornecer uma maneira prática e fácil de realizar experimentos em Sistemas de Recomendação. Nesse aspecto, o arquivo de configuração criado para o projeto permitiu mapear todas as informações necessárias para ser possível executar um experimento de ponta-a-ponta, as chaves e valores criados foram suficientes para que todo processo fosse realizado. Assim, garante-se que, a partir desse arquivo, todas as informações relacionadas a classes, configuração do *deploy* (informações do *cluster Kubernetes* para utilização do **Xperimentor**) e outras necessidades fossem atendidas. Com isso, obtém-se um ponto de partida inicial para o *framework*, capaz de centralizar todas as informações necessárias para execução. Essa característica facilita as configurações iniciais, por não ser necessário gerenciar diferentes arquivos com propósitos diferentes, por outro lado, esse arquivo pode crescer muito de acordo com a quantidade recursos que serão utilizados e também quantos experimentos serão definidos em um mesmo arquivo de configuração. Esse ponto pode gerar um desconforto na perspectiva de análise e visualização dessas informações definidas.

Outro resultado importante obtido no desenvolvimento desse trabalho, foram as documentações criadas para entendimento do **RecSysExp**, bem como os projetos que o complementam. Quando o desenvolvimento do *framework* foi iniciado, os projetos relacionados a ele, como o **Xperimentor** e **RecMetrics**, não possuíam uma documentação inserida em seus repositórios do **Github**. A única forma de documentação existente eram os textos das monografias dos autores. Esse fato acabou levando a um esforço maior do que o necessário para o entendimento inicial desses projetos, dessa forma, um dos objetivos aqui propostos foi criar uma documentação que permitisse um entendimento mais simples do *framework* e seus recursos. Dito isso, no repositório do *framework* conta-se com um conjunto de documentações escritas em arquivos *readme.md* que fazem explicações sobre a arquitetura do projeto, suas dependências, funcionalidades, como contribuir, como executar o projeto e também informações sobre os trabalhos relacionados como, por exemplo, a documentação criada para o **RecMetrics** que descreve informações importantes sobre o projeto bem como explica como utilizar o seu arquivo de configuração. Toda essa documentação foi escrita em Português com o objetivo de simplificar o entendimento dos alunos e colaboradores do projeto, mesmo que descrever em inglês permitisse que projeto fosse acessível de forma universal. A ideia foi permitir que nesse estágio inicial do desenvolvimento do **RecSysExp** o entendimento da documentação fosse mais simples considerando que vários utilizadores pudessem não ter um bom domínio da língua. Outra forma forma de documentação que foi inserida é a utilização de *docstrings* do Python, essa forma de documentar permite que ao avaliar e utilizar o código fonte do projeto seja possível ter de fácil acesso a descrição do que um método faz e também quais são os parâmetros esperados por ele. Da definição de *docstrings* é possível usar geradores de documentação capazes de entender e renderizar o que foi escrito e definido para todos os métodos que usem desse recurso.

O próximo ponto de observação pode ser as estruturas definidas para o projeto, pensando tanto em organização do repositório, definição de classes e métodos, armazenamento de resultados e por ai em diante. Sobre a organização do projeto, o que se destaca aqui é a criação de área especificas para cada necessidade envolvida no uso do **RecSysExp**, ou seja, além de considerar as estruturas de armazenamento de código fonte e testes do *framework* foi de grande valia criar diretórios onde exemplos de uso do projeto fossem facilmente encontrados. Foi criado também um diretório no projeto capaz de gerenciar e armazenar todos os resultados obtidos em cada etapa executada pelo *framework*, a forma de armazenamento desses resultados está descrita na Seção 3.15 e o principal ponto positivo dessa estrutura é que, garantindo o armazenamento dos resultados, conseguimos que o experimentador tenha conhecimento de maneira organizada dos artefatos gerados pelo experimento dele, além disso, outra possibilidade criada é a reutilização desses resultados seja fazendo uso deles para gerar algum tipo de análise a parte ou até mesmo pensando que caso uma execução do experimento dê algum problema, no futuro seja possível retomar a execução do ponto onde a execução foi interrompida. Outra estrutura importante, criada como uma necessidade da disciplina de BCC409, foi uma área para armazenamento de contribuições acadêmicas. Essas contribuições estão armazenadas na pasta *academic*, disponível na raiz do projeto, que tem como objetivo receber artefatos criados através de disciplinas ou outras atividades acadêmicas, definindo melhor o objetivo, queremos que ao longo do tempo essa área seja alimentada em vários períodos letivos em que a disciplina de Sistemas de Recomendação for ministrada com trabalhos práticos desenvolvidos pelos alunos.

A respeito da definição de classes e métodos, foi obtido como resultado a implementação de estruturas de classe que garantirão a evolução do projeto na perspectiva de especialização. Então, apesar de determinadas abordagens ainda não estarem implementadas de maneira concreta no *framework*, o conjunto de métodos base que permitem essa especialização existem, e foram esboçados no Capitulo 3, lembrando que diferentes recursos foram gerados para cada módulo, dentre eles: carregamento, transformação e armazenamento de diferentes formatos de arquivos através de uma classe **Loader**, disponibilização da base de dados MovieLens em diferentes dimensões, funções para garantir o download das bases de dados caso não existam, funções de pré-processamento referentes a normalização, codificação de recursos, divisão dos dados através de diferentes estratégias, processamento textual considerando diferentes técnicas, diferentes algoritmos de recomendação com métodos para treinamento, predição e recomendação, métricas voltadas para avaliação das predições e recomendações e formas de visualização com foco em gráficos de barra e dispersão. Além dessa definição inicial das estruturas de classes, recursos utilitários também foram criados para garantir processos como a execução de arquivos externos, *deploy* de aplicações, padronização de arquivos de configuração em diferentes formatos, limpeza dos experimentos, criação de pastas que não existem, dentre outras funcionalidades.

Tendo apresentado os resultados da perspectiva do desenvolvimento do *framework*, é necessário avaliar os resultados dos experimentos definidos. Relembrando, a ideia foi usar a base de dados MovieLens e avaliar diferentes amostras dos dados, com isso, foram definidos quatro

dimensões diferentes para a base: 100 mil, 200 mil, 300 mil e 400 mil registros. Para cada uma dessas bases foram usados os algoritmos **ItemKNN**, **UserKNN**, **Bias** e **BiasedSVD** avaliados com as métricas: **RMSE**, **NDCG**, **MAE**. Os resultados estão listados nas tabelas abaixo:

Tabela 4.1 – Resultados para o Experimento 1

Experimento 1 - MovieLens 100k			
Recomendadores	RMSE	NDCG	MAE
ItemKNN	1.458033	0.058862	1.161295
UserKNN	1.375486	0.062897	1.109742
Bias	1.344586	7.274982	1.095435
BiasedSVD	1.360211	6.960768	1.102076

Tabela 4.2 – Resultados para o Experimento 2

Experimento 2 - MovieLens 200k			
Recomendadores	RMSE	NDCG	MAE
ItemKNN	1.781070	7.953035	1.443856
UserKNN	1.619052	5.715714	1.296853
Bias	1.382936	7.541084	1.084114
BiasedSVD	1.412329	7.970832	1.106677

Tabela 4.3 – Resultados para o Experimento 3

Experimento 3 - MovieLens 300k			
Recomendadores	RMSE	NDCG	MAE
ItemKNN	1.811657	8.079413	1.469538
UserKNN	1.643986	6.394207	1.324030
Bias	1.361154	8.976651	1.068685
BiasedSVD	1.402332	8.032357	1.099792

Tabela 4.4 – Resultados para o Experimento 4

Experimento 4 - MovieLens 400k			
Recomendadores	RMSE	NDCG	MAE
ItemKNN	1.814016	8.146988	1.468198
UserKNN	1.640548	6.672966	1.321942
Bias	1.344889	8.447236	1.056532
BiasedSVD	1.389991	7.903718	1.091164

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, vamos considerar 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. Sabendo disso, podemos avaliar algumas das visualizações abaixo:

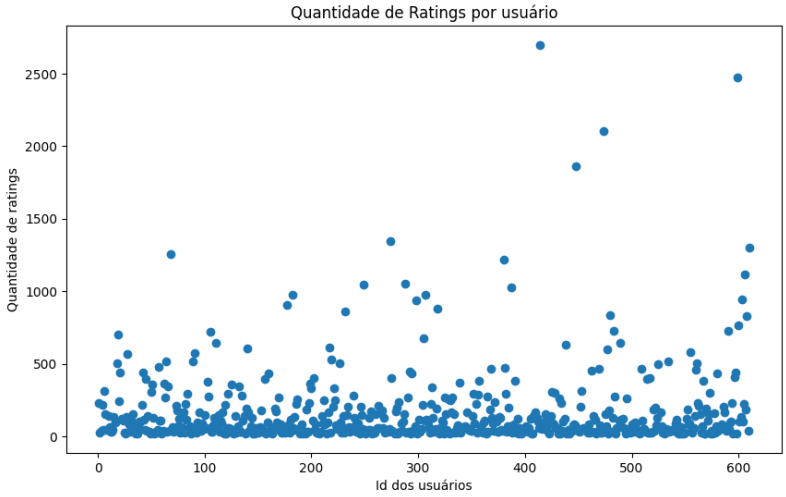

Figura 4.1 – Quantidade de Ratings por Usuário - Visualização em gráfico de dispersão.

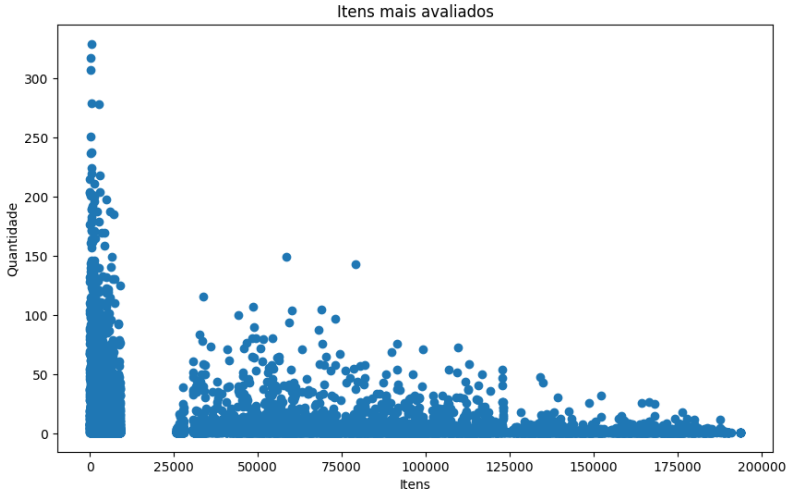

Figura 4.2 – Itens mais avaliados na base MovieLens100K.

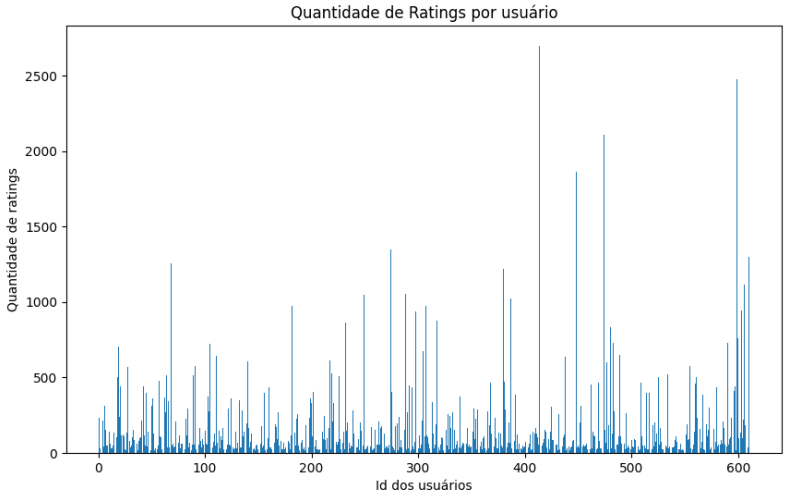

Figura 4.3 – Quantidade de Ratings por Usuário - Visualização em gráfico de barras.

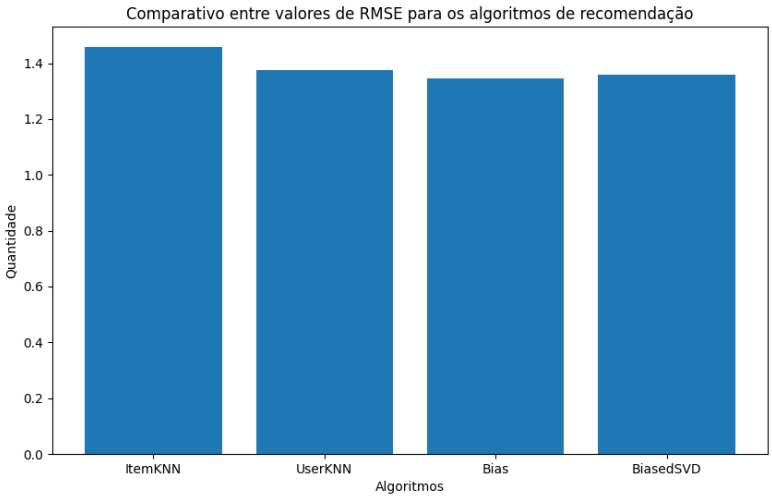

Figura 4.4 – Comparativo entre valores de RMSE para os algoritmos de recomendação.

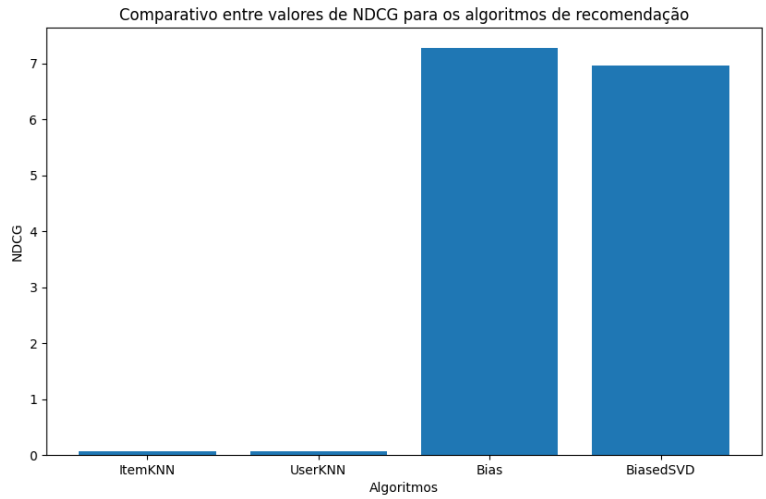

Figura 4.5 – Comparativo entre valores de NDCG para os algoritmos de recomendação.

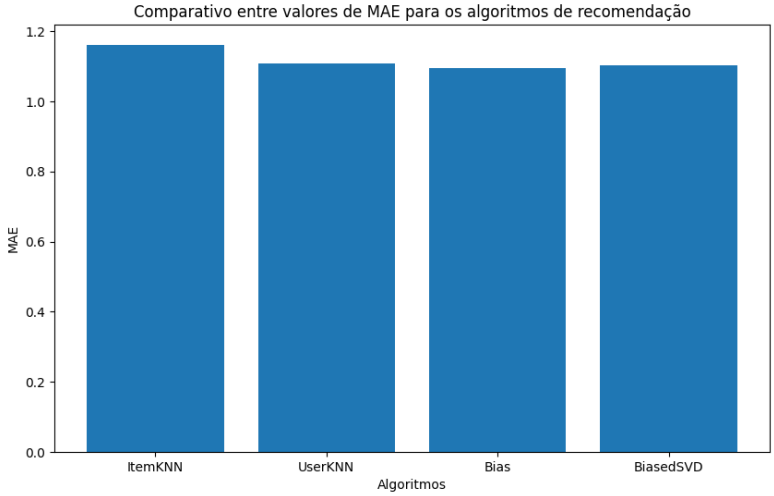

Figura 4.6 – Comparativo entre valores de NDCG para os algoritmos de recomendação.

Todas essas visualizações são geradas através de arquivos de extensão **png**, e são armazenadas de acordo com sua categoria, segmentando inicialmente se esse gráfico gerado é estático ou interativo e posteriormente o armazenando em diretórios que representam o seu tipo (barra, dispersão, pizza, etc). Portanto, nessa seção apresentamos os resultados gerados ao longo do desenvolvimento e teste do *framework*, a seguir na Seção 4.2 discutiremos sobre estudos de caso que foram feitos na disciplina de BCC409 apresentando as dificuldades, observações e conclusões baseadas na observação do uso do **RecSysExp**.

4.2 Estudos de Caso

Este *framework* foi submetido a alguns estudos de caso ao longo de seu desenvolvimento, durante a oferta da disciplina de BCC409 (Sistemas de Recomendação), do DECOM-UFOP, ministrada pelo professor Reinaldo Silva Fortes. Com isso, possibilitou-se a descoberta de problemas e necessidades que serviram de base para a realização de melhorias e também tomadas de decisão. Inicialmente, o primeiro estudo feito foi a utilização do projeto em diferentes ambientes de desenvolvimento com o intuito de descobrir problemas relacionados a dependências, por exemplo. Para a realização desse estudo os alunos precisaram utilizar em suas máquinas o **RecSysExp** e nesse intuito foi necessário que todos fizessem o *clone* do projeto, mantido no **Github**, e que o ambiente virtual do Python fosse montado a partir do arquivo *requirements.txt*. No momento que esse processo foi feito, começaram a aparecer os primeiros desafios relacionados às dependências e também aos sistemas operacionais, pois cada aluno usava um tipo de sistema operacional, arquitetura e até mesmo versões do Python diferentes.

Desse primeiro estudo, realizado nas máquinas dos alunos, foi possível perceber que a utilização do sistema operacional **Linux** facilitava o uso do *framework*, isso porque proporcionava menos erros de compatibilidade, por exemplo. Ao ser utilizado no sistema operacional **MAC**, vários problemas relacionados às dependências foram aparecendo, impossibilitando a utilização do *framework*. Problemas similares, com as dependências, também foram encontrados no **Windows**, lembrando que, para uma melhor experiência de uso e instalação dos recursos nesse sistema operacional, foi necessário que os alunos optassem pelo uso do **WSL** (Subsistema do Windows para Linux).

Foi possível constatar que as dependências eram o primeiro desafio, considerando que um ambiente estável deveria existir para que todos os alunos pudessem utilizar com tranquilidade o *framework*. A maior parte dos problemas se dava por incompatibilidade entre versões, ou seja, determinadas bibliotecas precisavam que outras bibliotecas estivessem uma faixa de versões especifica, sendo assim, foi necessário definir estruturas condicionais no arquivo *requirements.txt*, para garantir que determinadas dependências estivessem sempre entre uma faixa de versões aceitáveis.

Outro problema notado durante a utilização do **RecSysExp**, foi o fato de existirem muitos arquivos envolvidos no projeto, e pelo tamanho de algumas dependências como, por exemplo, o **TensorFlow**. Além dos arquivos criados no *framework*, existiam os projetos do **Xperimentor**, **RecMetrics** e o doutorado do Professor Reinaldo Silva Fortes inseridos no repositório principal. Isso causou uma demora no processo de clonagem do projeto e também no *download* e instalação das dependências. Então, para garantir uma experiência mais rápida e fluida, foi necessário remover alguns desses projetos para diminuir a quantidade de recursos que seria obtido através do repositório do **Github**. O projeto do **RecMetrics** existia em duas formas: um arquivo *.jar*, que permitia sua execução e um projeto com o código fonte desse *.jar*, dessa forma, somente o arquivo *jar* foi mantido visto que a partir dele conseguimos obter todos os resultados que

precisamos, esse arquivo está disponível para uso no repositório do **Github**. O código fonte do doutorado do Professor Reinaldo Silva Fortes foi inserido na estrutura do projeto principalmente pela necessidade de entendimento e abstração das funcionalidades nele contidas, portanto, como ele estava sendo usado apenas para referencial teórico optamos por removê-lo da estrutura do *framework*. O projeto que foi mantido na estrutura do **RecSysExp** foi o **Xperimentor**, isso porque o seu *back-end* (TaskExecutor) é usado diretamente para a execução das tarefas e o seu *front-end* poderá no futuro ser construído, instanciado e executado automaticamente pelo *framework*.

Após a descoberta e correção de problemas relacionados às dependências do *framework*, o próximo desafio foi relacionado a garantir que todos os grupos da disciplina conseguissem estar preparados e contextualizados das características, recursos e de como utilizar o projeto para que pudessem contribuir com o **RecSysExp** no **Github**. O tempo escasso e a curva de aprendizado longa foram empecilhos na hora da utilização, sabemos que é natural os alunos precisarem de um tempo razoável para se inserirem em uma nova tecnologia, isto porque muitos deles possuem várias outras demandas, tanto na Universidade quanto na vida profissional e pessoal. Além disso, a maioria se não todos os alunos estavam tendo o primeiro contato com a área de Sistemas de Recomendação e por ser um universo de estudo muito vasto, absorver esse grande volume de novas informações e precisar conhecer um projeto em desenvolvimento gerou um grande dificultador para a utilização do mesmo de forma eficiente. Com isso, um detalhe que se mostrou crucial para melhorar o entendimento dos alunos perante o *framework* foi acompanhar de perto o trabalho dos grupos, para mostrar como eles poderiam executar as tarefas que precisariam e também instruí-los a como estender as classes além de mostrar onde certos recursos poderiam ser encontrados.

Ao longo das aulas de laboratório propostas na disciplina foi possível perceber que os exemplos de utilização do **RecSysExp**, contidos na pasta *examples* do repositório, foram de grande ajuda no entendimento do trabalho, pois, a partir desses exemplos, os alunos conseguiram entender como fazer todo o processo de recomendação. Esses exemplos tinham foco principal em instanciar uma base de dados, aplicar algum pré-processamento e posteriormente realizar a predição dos *ratings* e recomendação dos itens. A partir desse entendimento foi possível concluir que pode ser mais interessante investir em documentações voltadas à parte prática (como utilizar determinados recursos) do que documentações mais teóricas.

Durante a disciplina, muitos alunos foram prejudicados na realização de seu trabalho prático por não conseguirem utilizar o **RecSysExp** nas máquinas do laboratório onde ocorriam as aulas práticas. A intenção inicial era que todos os membros da disciplina pudessem utilizar dessas máquinas para fazer uso do projeto. No entanto, para que isso fosse possível seria necessário que todas as máquinas estivessem devidamente pré-configuradas com o *framework* e suas dependências. Garantir esse cenário era algo totalmente dependente do técnico responsável pelo laboratório e, como esse não foi um aspecto previamente considerado, grande parte dos alunos não levava seu próprio notebook para as aulas. Dessa forma, os discentes ficavam impossibilitados de progredir

no desenvolvimento do seu trabalho prático caso não optassem por fazer uso do *Google Colab* (alternativa que permitia a execução do projeto). Para o caso dos alunos que optavam pelo uso do *Colab*, eles conseguiam fazer a execução das funcionalidades desenvolvidas, porém, a plataforma não fornece uma interface adequada para contribuição no código fonte como acontece em uma IDE (do inglês, Integrated Development Environment), por exemplo.

Nesses estudos de caso feitos, foi possível observar um conjunto de ações necessárias para a evolução do projeto e também para seu uso em diferentes cenários. Então, apesar dessas observações, não estarem estruturadas através de um formulário, por exemplo, o que foi observado serve de referencial para tomada de decisão, melhorias e correções. Pensando inicialmente na disciplina ofertada, sabemos que os alunos conseguiram se adaptar e entender melhor o *framework* a partir de exemplos práticos de uso. Mesmo existindo vídeos (seminários em BCC409) explicando o funcionamento dos recursos e também exemplificando como utilizá-los, a forma que pareceu ser a preferência dos alunos foi existirem arquivos no projeto que realizavam alguma ação especifica de determinado módulo do trabalho como, por exemplo, aplicação de pré-processamentos na base de dados, recomendação de itens, especialização de classes e por ai em diante. Então, uma abordagem mais prática seja na documentação ou na explicação dos recursos precisa ser o foco durante a evolução do projeto. Além disso, mesmo existindo os comentários a nível de código que podem ser renderizados na IDE, a ausência de um documentação *online* parece ter sido um dificultador. Por fim, a disciplina de BCC409, ministrada no período de 2022/2, permitiu que cinco grupos fizessem contribuições ao projeto através da realização de um trabalho prático que envolvia sistemas de recomendação. Nesses resultados, cada grupo utilizou de recursos definidos no *framework* para realizar seus experimentos e de posse do que foi gerado por cada grupo e de acordo com as dificuldades encontradas, outros trabalhos conseguirão utilizar desses artefatos para evoluir de maneira direcionada. Para o caso do período de 2022/2, várias temáticas foram propostas e serão inseridas no **RecSysExp** na área *academic*, mencionada na Seção 4.1, algumas delas são:

1. Sistemas de Recomendação para Música: uma abordagem focada em *Fairness*
2. Sistemas de Recomendação de Música para dois usuários
3. Sistemas de Recomendação para jogos da *Steam*
4. Realização de experimentos em Sistemas de Recomendação para uma base de dados do domínio de livros
5. Realização de experimentos para avaliar estratégias de validação cruzada

Os temas definidos para esse período permitiram que o *framework* fosse utilizado em diferentes problemas. Cada temática tinha diferentes necessidades como, a inserção de novas bases de dados seja para música, livros (Book-Crossing) e jogos da Steam, utilização de funções

de pré-processamento, funções de otimização, dentre outras funcionalidades. A inserção de bases de dados como a do **Book-Crossing**, por exemplo, mostrou que os métodos elencados para representar os itens, usuários e avaliação foram úteis e replicáveis na representação das diferentes bases de dados do Book-Crossing (diferentes arquivos CSV que representam usuários, itens e avaliações). Da utilização de funções de pré-processamentos, as mais utilizadas e validadas pelos alunos foram relacionadas à normalização, cálculo do *Term Frequency-Inverse Document Frequency* (TF-IDF), divisão da base em *folds* e em amostras de treino e teste. A normalização foi muito utilizada porque, considerando a existência de avaliações em diferentes escalas numéricas, a normalização ajuda na padronização dos resultados, viabilizando análises comparativas. Sobre o cálculo do **TF-IDF**, seu uso se deu principalmente pela necessidade de alguns grupos trabalharem com recursos textuais. Em relação à divisão da base de dados em *folds*, os recursos implementados no *framework* foram úteis para o grupo que estava realizando experimentos relacionados à validação cruzada. No **RecSysExp** foi desenvolvida uma classe para lidar com os *folds*, que usa o *pattern* Strategy, permitindo que diferentes estratégias para divisão sejam feitas usando recursos da biblioteca **Sklearn** como: **KFold**, **StratifiedKFold**, **GroupKFold**, dentre outros. Essa base previamente criada forneceu insumos para que os experimentos pudessem ser feitos sem grandes esforços de implementação. Durante o desenvolvimento desses trabalhos também foram inseridas necessidades que ainda não estão presentes no *framework*, mas que servirão, principalmente, pensando em trabalhos futuros. Um exemplo é a abordagem voltada para *Fairness*, discutida por Li et al. (2022) em seu trabalho e a necessidade de um grupo para implementar técnicas de otimização para aplicar em seu problema. Ambas as necessidades são temáticas importantes, pensando no estudo de Sistemas de Recomendação.

Sendo assim, foram apresentados todos os resultados obtidos durante o desenvolvimento do projeto e, no Capítulo 5, será apresentado algumas considerações finais sobre o *framework* e também serão apresentadas sugestões de trabalhos futuros.

5 Considerações Finais

Em resumo, este trabalho teve como objetivo propor um *framework* para a implementação e validação de sistemas de recomendação. Além disso, era o intuito fornecer um conjunto de interfaces e classes que permitissem a generalização e especialização de novas classes, de forma a permitir que o projeto cresça de maneira desacoplada e com poucos problemas. Essas características visam criar um cenário no qual podemos introduzir recursos de terceiros e próprios, de forma a compor o *framework* de forma mais completa possível.

Como objetivos específicos elencamos que o *framework* pudesse incluir novos paradigmas relacionados a sistemas de recomendação, proporcionasse facilidade para experimentação e base para evolução de um trabalho voltado a gestão, execução e reprodução de experimentos em Sistemas de Recomendação. Durante esse processo, atingimos uma revisão bibliográfica extensa sobre os conteúdo de recomendação, suas abordagens e também os trabalhos que compõe base para esta monografia. Após todo esse estudo, foi possível elencar os requisitos obrigatórios e quais seriam os pontos que proporcionariam a concepção desse *framework* bem sucedida.

Conhecendo as necessidades e requisitos, uma visão geral da arquitetura foi proposta, bem como a especialização dos componentes definidos. O detalhamento de cada uma das partes foi feito inicialmente considerando uma visão mais alto nível dos módulos, classes e interfaces. Ao longo do desenvolvimento foi possível conhecer novos requisitos e também problemas a serem resolvidos. Muito do que foi descoberto foi devido a utilização do projeto por várias pessoas na disciplina de BCC409, esse fato evidenciou a importância de manter uma rotina constante de validação do *framework*.

Como esse trabalho se trata de um software, sabemos que não existe um fim para sua evolução, podemos inserir novas funcionalidades, podemos corrigir *bugs*, refatorar módulos e por ai em diante. Ainda assim, dado o escopo definido neste trabalho que previa uma abordagem *end-to-end*, conseguimos garantir uma entrega que englobasse todos os módulos previstos no escopo do trabalho incluindo: bases de dados, pré-processamentos, recomendação, avaliação e visualização. Podemos enxergar essa entrega com um mínimo produto viável, imaginando o potencial e tamanho que o **RecSysExp** pode atingir, a partir de agora vamos conseguir visualizar um conjunto de próximos passos para esse trabalho e eles serão descritos a seguir na Seção 5.1.

5.1 Trabalhos Futuros

Conhecendo os resultados que obtivemos até então, conseguimos vislumbrar os próximos passos pensando em sua evolução. Uma das possíveis ações é transformar toda a documentação desse projeto para a língua inglesa, garantindo que sua utilização possa ser feita por qualquer um

em qualquer parte do mundo. Outro ponto relacionado a documentação é que o projeto precisa ter sua documentação de código (*docstrings*) renderizada na Web através de alguma ferramenta de geração de documentação como, Sphinx, por exemplo. Isso garantirá uma maior facilidade no entendimento de todos os recursos existentes, até porque existem muitos recursos que não estão inseridos dentro de alguns dos módulos principais, mas que são de grande valia para outros processos (funções utilitárias). Então, com uma documentação como essa, seria possível mostrar toda e qualquer função documentada através das *docstrings*.

Outra ação necessária para o **RecSysExp** é buscar uma forma de gerenciar esse projeto em um repositório de código que permita que vários colaboradores interajam com o projeto, contando com a definição de *issues*, discussões, gerenciamento de equipes, pipelines de testes, integração contínua, dentre outras funcionalidades. Apesar desse repositório já existir e permitir a colaboração entre várias pessoas, existem várias características que podem ser inseridas no gerenciamento garantindo que esse projeto seja um *open-source* bem sucedido, algumas dessas características não foram abordadas devido a restrições que repositórios pessoais tem dentro da plataforma do **Github**.

Pensando a nível de código, poderíamos fazer uma série de melhorias e evoluções no projeto atual, uma dessas evoluções seria inserir em vários módulos classes concretas genéricas que possam garantir uma implementação padronizada de algum novo recurso sem que necessariamente seja preciso criar uma nova classe para isso. Ou seja, considerando as bases de dados, ao invés de implementar uma nova classe para toda nova base de dados que for utilizada, poderia existir uma classe **GenericDataset** capaz de ser instanciada e usada com propósitos gerais. Essa necessidade existe, já que não podemos instanciar classes abstratas, como o caso de **AbstractDataset**. Ainda em relação à implementação, como um trabalho futuro o **RecSysExp** pode aprofundar em questões relacionadas à Hibridização. Ou seja, implementar os diferentes *design's* existentes e também fazer testes comparativos entre os resultados das CF e CB em relação à hibridização, avaliar mais a fundo os resultados da inserção das *meta-features* nos modelos de recomendação, visto que esse é um dos focos da linha de pesquisa do Professor Reinaldo Silva Fortes. Além disso, voltar ao foco a outros tipos de filtragem, como a demográfica e multi-objetivo e buscar formas robustas de avaliar, considerando diferentes critérios como acurácia, novidade e diversidade.

Também é possível que um trabalho em relação à visualização dos dados seja realizado. Desse conjunto de resultados gerados durante o experimento, seria possível desenvolver *dashboards* para condensar as análises em uma única visualização, garantindo mais praticidade na hora de interpretar os resultados do experimento, tendo em vista que, no cenário atual, trabalhamos com imagens criadas separadamente para cada tipo de forma de visualização solicitada, ou seja, se for solicitada a criação de visualizações que permitam comparar os resultados das métricas **RMSE** e **MAE** em alguns algoritmos de recomendação, serão gerados duas imagens, uma para cada métrica. Portanto, com a inserção dos *dashboards* será possível evitar a criação dessas

visualizações separadas, condensando em uma única estrutura todas as visualizações solicitadas. No geral, podemos pensar em diferentes especializações envolvendo cada um dos módulos, por exemplo, os experimentos e desenvolvimento do trabalho foram voltados inicialmente para a base de dados, **MovieLens**, então para o futuro seria interessante incluir diversas outras bases nesse projeto, de forma que os experimentos possam ser feitos considerando diferentes contextos.

Outro trabalho que ainda não foi realizado nessa etapa do desenvolvimento, é garantir que o *framework* possua uma grande cobertura de testes, sejam eles unitários ou de integração. Além disso, podemos criar formas de mensurar o desempenho do **RecSysExp**, seja avaliando o tempo de treinamento dos modelos, quantidade de uso de CPU e RAM nos processos, dentre outras mensurações.

Pensando na execução do projeto, atualmente ele permite que a execução seja feita a partir de uma linha de comando. No entanto, como um trabalho futuro seria interessante que todo esse processo de experimentação em sistemas de recomendação pudesse ser realizado a partir de uma interface *Web* com as características necessárias para a realização do experimento.

Esses foram alguns exemplos de melhoria para o **RecSysExp**, o escopo para sua evolução é grande e diverso e, no futuro, pontos de melhoria estarão listados através do repositório do **Github**.

Referências

ADOMAVICIUS, G.; ZHANG, J. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems (TMIS)*, ACM New York, NY, USA, v. 3, n. 1, p. 1–17, 2012.

ANELLI, V. W.; BELLOGIN, 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: . New York, NY, USA: Association for Computing Machinery, 2021. (SIGIR '21), p. 2405–2414. ISBN 9781450380379. Disponível em: <https://doi.org/10.1145/3404835.3463245>.

BIAS. *Bias*. Bias, 2019. Disponível em: <https://lkpy.readthedocs.io/en/stable/bias.html#lenskit.algorithms.bias.Bias>.

BIASEDMF. *Classic Matrix Factorization*. BiasedMF, 2019. Disponível em: <https://lkpy.readthedocs.io/en/stable/mf.html#module-lenskit.algorithms.svd>.

BOBADILLA, J.; ORTEGA, F.; HERNANDO, A.; GUTIéRREZ, A. Recommender systems survey. *Knowledge-Based Systems*, v. 46, p. 109–132, 2013. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705113001044>.

BRUCKNER, C. H. P. *Framework para extração de meta-features para bases de dados de sistemas de recomendação*. 45 f. Monografia (Monografia) — Universidade Federal de Ouro Preto, 2017.

BURKE, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, Springer, v. 12, n. 4, p. 331–370, 2002.

BURKE, R. Hybrid web recommender systems. *The adaptive web*, Springer, p. 377–408, 2007.

DANGETI, P. *Statistics for Machine Learning*. Packt Publishing, 2022. Disponível em: <https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/682d8506-6969-46a2-bc3f-fabf7015b59c.xhtml>.

EKSTRAND, M. D. The LKPY package for recommender systems experiments: Next-generation tools and lessons learned from the lenskit project. *CoRR*, abs/1809.03125, 2018. Disponível em: <http://arxiv.org/abs/1809.03125>.

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 (Doutorado) — Universidade Federal de Ouro Preto, 2022. Unpublished thesis.

FORTES, R. S.; FREITAS, A. R. de; GONÇALVES, M. A. A multicriteria evaluation of hybrid recommender systems: on the usefulness of input data characteristics. In: SCITEPRESS. *International Conference on Enterprise Information Systems*. [S.l.], 2017. v. 2, p. 623–633.

FORTES, R. S.; LACERDA, A.; FREITAS, A.; BRUCKNER, C.; COELHO, D.; GONCALVES, M. User-oriented objective prioritization for meta-featured multi-objective recommender systems.

In: *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*. New York, NY, USA: Association for Computing Machinery, 2018. (UMAP '18), p. 311–316. ISBN 9781450357845. Disponível em: <https://doi.org/10.1145/3213586.3225243>.

GOOGLE. *Content-based filtering advantages and disadvantages; recommendation systems; google developers*. Google, 2018. Disponível em: <https://developers.google.com/machine-learning/recommendation/content-based/summary>.

GOOGLE. *Collaborative Filtering Advantages and disadvantages - recommendation systems - google developers*. Google, 2020. Disponível em: <https://developers.google.com/machine-learning/recommendation/collaborative/summary>.

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.

HUG, N. Surprise: A python library for recommender systems. *Journal of Open Source Software*, The Open Journal, v. 5, n. 52, p. 2174, 2020. Disponível em: <https://doi.org/10.21105/joss.02174>.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

HURLEY, N.; RICKARD, S. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, IEEE, v. 55, n. 10, p. 4723–4741, 2009.

IMPLICITMF. *Classic Matrix Factorization*. ImplicitMF, 2019. Disponível em: <https://lkpy.readthedocs.io/en/stable/mf.html#lenskit.algorithms.als.ImplicitMF>.

ITEMKNN. *k-NN Collaborative Filtering*. ItemKNN, 2019. Disponível em: <https://lkpy.readthedocs.io/en/stable/knn.html#lenskit.algorithms.item_knn.ItemItem>.

JUNG, S. Y.; HONG, J.-H.; KIM, T.-S. A statistical model for user preference. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 17, n. 6, p. 834–843, 2005.

LANG, K. Newsweeder: Learning to filter netnews. In: *Machine Learning Proceedings 1995*. [S.l.]: Elsevier, 1995. p. 331–339.

LI, Y.; CHEN, H.; XU, S.; GE, Y.; TAN, J.; LIU, S.; ZHANG, Y. *Fairness in Recommendation: A Survey*. 2022.

METEREN, R. V.; SOMEREN, M. V. Using content-based filtering for recommendation. In: *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*. [S.l.: s.n.], 2000. v. 30, p. 47–56.

NEPOMUCENO, F. A. R. *Uma análise experimental sobre sistemas de filtragem colaborativa*. 48 f. Monografia (Monografia) — Universidade Federal de Ouro Preto, 2014.

PACHECO, M. *Xperimentor: Um framework para gerenciamento de execução de experimentos computacionais*. 51 f. Monografia (Monografia) — Universidade Federal de Ouro Preto, 2019.

PAN, R.; ZHOU, Y.; CAO, B.; LIU, N. N.; LUKOSE, R.; SCHOLZ, M.; YANG, Q. One-class collaborative filtering. In: *2008 Eighth IEEE International Conference on Data Mining*. [S.l.: s.n.], 2008. p. 502–511.

PASTOR, G.; MORA-JIMÉNEZ, I.; JÄNTTI, R.; CAAMANO, A. J. Mathematics of sparsity and entropy: Axioms core functions and sparse recovery. *arXiv preprint arXiv:1501.05126*, Jan, 2015.

RESNICK, P.; VARIAN, H. R. Recommender systems. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 40, n. 3, p. 56–58, mar 1997. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/245108.245121>.

SALAH, A.; TRUONG, Q.-T.; LAUW, H. W. Cornac: A comparative framework for multimodal recommender systems. *Journal of Machine Learning Research*, v. 21, n. 95, p. 1–5, 2020. Disponível em: <http://jmlr.org/papers/v21/19-805.html>.

SALTER, J.; ANTONOPOULOS, N. Cinemascreen recommender agent: combining collaborative and content-based filtering. *IEEE Intelligent Systems*, IEEE, v. 21, n. 1, p. 35–41, 2006.

SCHAFER, J. B.; FRANKOWSKI, D.; HERLOCKER, J.; SEN, S. Collaborative filtering recommender systems. In: ____. *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 291–324. ISBN 978-3-540-72079-9. Disponível em: <https://doi.org/10.1007/978-3-540-72079-9_9>.

SELIM, S.; SAHAL, R.; ELKORANY, A. *An Adaptive Framework for Enhancing Recommendation Using Hybrid Techniques*. 2014.

SOUZA, A. N. de Paula e. *Um framework para o cálculo de métricas de caracterização de dados de filtragem colaborativa*. 60 f. Monografia (Monografia) — Universidade Federal de Ouro Preto, 2014.

USERKNN. *k-NN Collaborative Filtering*. UserKNN, 2019. Disponível em: <https://lkpy.readthedocs.io/en/stable/knn.html#lenskit.algorithms.user_knn.UserUser>.

VARGAS, S.; CASTELLS, P. Rank and relevance in novelty and diversity metrics for recommender systems. In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2011. (RecSys '11), p. 109–116. ISBN 9781450306836. Disponível em: <https://doi.org/10.1145/2043932.2043955>.

WASKOM, M. L. seaborn: statistical data visualization. *Journal of Open Source Software*, The Open Journal, v. 6, n. 60, p. 3021, 2021. Disponível em: <https://doi.org/10.21105/joss.03021>.

XU, Y.; LI, Y.; JOSONG, A. Recommender systems for web intelligence. *J. Emerg. Technol. Web Intell*, v. 2, n. 4, p. 269, 2010.