UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA

PAULO ROBERTO XAVIER JÚNIOR

ContextCF.jl: Um Framework para CARS em Filtragem Colaborativa

Prof. Filipe Braida do Carmo, D.Sc.
Orientador

Rio de Janeiro, Dezembro de 2018

ContextCF.jl: Um Framework para CARS em Filtragem Colaborativa

Paulo Roberto Xavier Júnior

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por:

_____

Paulo Roberto Xavier Júnior

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Bruno José Dembogurski, D.Sc.

_____

Prof. Julio César Barbieri Gonzalez de Almeida, M.Sc.

RIO DE JANEIRO, RJ - BRASIL

Dezembro de 2018

Agradecimentos

Gostaria de agradecer primeiramente ao corpo de docentes do curso de ciência da computação da Universidade Federal Rural do Rio de Janeiro, por terem propiciado um curso de altíssima qualidade, em especial ao professor Filipe Braida, por sua amizade, dedicação, paciência e suporte acima de qualquer expectativa ou obrigação, além das diversas lições profissionais e pessoais que foram tiradas durante todo o curso, de Computação 1 ao Projeto Final.

Gostaria de agradecer à minha mãe Dejanira, que forneceu apoio integral e teve fé no meu crescimento pessoal desde meu nascimento, sendo o maior exemplo de honestidade e resiliência que tive em minha vida. À minha tia, Marlene, por suas palavras doces, suporte e apoio durante todos os anos de curso. Ao meu grande amigo e irmão Maurício Alves, por ter me dado suporte pela última década, acreditando no meu crescimento com uma fé inabalável e tendo sido imprescindível nessa jornada. À minha namorada Carolina, e seu suporte, compreensão e incentivo, tornando esse fardo mais leve.

Gostaria de agradecer ao meu amigo Gabriel Segobia, por ter estado comigo integralmente durante a graduação, dividido todas as dificuldades e ter sido o suporte nas horas mais necessárias, tendo tornado essa graduação possível. Gostaria de agradecer também ao meu amigo Vitor Machado que além de um grande mentor no trabalho, tendo colaborado no meu desenvolvimento, ajudou na revisão desse texto diversas vezes.

E por último, gostaria de agradecer a todas as forças que compõe esse universo por terem me permitido crescer, amadurecer e chegar até aqui.

RESUMO

ContextCF.jl: Um Framework para CARS em Filtragem Colaborativa

Paulo Roberto Xavier Júnior

Dezembro/2018

Orientador: Filipe Braida do Carmo, D.Sc.

Sistemas de recomendação são usados para fornecer informação filtrada de uma grande quantidade de elementos. Existem diferentes técnicas e algoritmos diferentes que podem ser usados em um recomendador, e sua escolha é feita conhecendo-se o domínio do problema e área a ser aplicada. Dentre as abordagens, os *Context-Aware Recommender Systems* (CARS) utilizam o contexto do usuário *e.g.* localização e clima, para oferecer recomendações de melhor qualidade aos usuários. Com a recente popularização dos sistemas de recomendação devido ao prêmio Netflix, e seu amplo uso por empresas como diferencial competitivo, o número de entusiastas e pesquisadores da área aumentou, gerando o surgimento de diversos *frameworks*. Nesse trabalho foram analisados os principais *frameworks* da área, e a partir das características comuns encontradas, foi construído um ecossistema para CARS, com o intuito de facilitar o aprendizado e uso das informações contextuais em sistemas de recomendação.

ABSTRACT

ContextCF.jl: Um Framework para CARS em Filtragem Colaborativa

Paulo Roberto Xavier Júnior

Dezembro/2018

Advisor: Filipe Braida do Carmo, D.Sc.

Recommender systems are used to provide filtered information of a large amount of elements. There are different techniques and algorithms that can be used in a recommendation, and their choice is made knowing the domain of the problem and the area to be applied. Among the approaches, the Context-Aware Recommender Systems (CARS) use the user context e.g. location and weather, to provide better quality recommendations to users. With the popularization of recommender systems due to the Netflix Prize and its wide use by companies as a competitive differential, the number of enthusiasts and researchers in the area increased, generating the emergence of many frameworks. In this paper, the main frameworks of the area were analyzed, and from the common characteristics found, an ecosystem was constructed for CARS, in order to facilitate learning and use of contextual information in recommender systems.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

SR	Sistema de Recomendação
CARS	Context Aware Recommender System
CDRS	Context-Driven Recommender System
JIT	Just in time
GC	Garbage Collector
LLVM	The LLVM Compiler Infrastructure
DSL	Domain Specific Language

Sumário

Capítulo 1

Introdução

O volume de informação produzido no mundo tem aumentado progressivamente, numa velocidade superior à nossa capacidade de assimilá-lo. O avanço das tecnologias de informação e comunicação popularizou o acesso a essas informações, ampliando as opções de conteúdos, produtos e serviços.

Devido a esse excesso de informações, encontrar conteúdo relevante tornou-se exaustivo. Com o intuito de auxiliar os usuários a lidar com a enorme quantidade de dados, os Sistemas de Recomendação (SR) despontaram como forma de oferecer recomendações personalizadas de seus conteúdos ou serviços. Empresas como *Spotify*[1], *Amazon*[2] e *Netflix*[3] tem esses sistemas como um de seus principais diferenciais competitivos.

Em Adomavicius e Tuzhilin (2015), um Sistema de Recomendação é definido como um conjunto de técnicas e ferramentas de software que fornecem sugestões de itens úteis para um usuário. Existem diversas abordagens para se efetuar a recomendação, sendo a filtragem colaborativa a mais popular delas. Nela, a recomendação é feita baseada em itens que usuários de perfil semelhante gostaram no passado.

Modelar interesses dos usuários para atender às necessidades individuais é um dos grandes desafios em aplicações de filtragem de informação e recomendação

[1] https://spotify.com
[2] https://amazon.com
[3] https://netflix.com

(BELLOGÍN, 2012). É uma combinação do que está ocorrendo ao redor do usuário no momento da interação com o sistema (situação) e o que ele está tentando realizar (intenção) (WHITE; BAILEY; CHEN, 2009). Contexto é definido como qualquer informação útil para caracterizar a situação de uma entidade e que pode afetar como os usuários interagem com o sistema (ABOWD et al., 1999).

Visando utilizar os dados contextuais e fornecer ao usuário informação relevante e/ou serviços baseados em seu contexto atual, surgiram os *Context-Aware Recommender Systems* (CARS). Na abordagem utilizada pelos CARS, a aplicação assume que a informação contextual é conhecida e definida, e que esses atributos contextuais afetam o resultado das recomendações. (ADOMAVICIUS; TUZHILIN, 2005).

Desde o prêmio *Netflix*, os SR se tornaram um tema popular e atrativo, pois foi disponibilizado um conjunto de dados de larga escala com 100 milhões de avaliações de filmes, atraindo milhares de estudantes, engenheiros e entusiastas do campo (ADOMAVICIUS; TUZHILIN, 2015). Com isso, diversos *frameworks* começaram a surgir, facilitando o estudo e promovendo fácil acesso para interessados no tema aos algoritmos clássicos de Sistemas de Recomendação.

O uso de informação contextual em SR melhora seu desempenho em relação aos que não usam essas informações (ADOMAVICIUS; TUZHILIN, 2005). Logo, a criação de *frameworks* que facilitem não só o estudo de SR, mas também de CARS seria relevante, dada sua importância. Nesse trabalho, foram analisados os principais *frameworks* da área, e foi notado que dentre os existentes, poucos dão suporte a informações contextuais.

Foram observadas algumas características comuns nos *frameworks* analisados, como a possibilidade da parametrização em suas funções de predição; fácil extensão, facilitando a construção de novos algoritmos ou de novos conjuntos de dados; presença de métricas tradicionais do domínio e em alguns casos, preocupação com desempenho e escalabilidade.

Com o intuito de facilitar e fomentar o estudo de CARS pela comunidade acadêmica, foi construído um ecossistema em código aberto de CARS que possui

as características desejáveis ao uso do cientista. É composto de dois módulos: *ContextCF.jl* que oferece uma camada de abstração para uso de informação contextual e *ContextDatasetsCF.jl*, módulo complementar que fornece algumas bases de dados de contexto para uso imediato.

1.1 Objetivo

Os principais objetivos do *ecossistema* são:

- Estender o *framework* Persa.jl, criando estruturas compatíveis com o mesmo e permitindo o uso de suas funcionalidades em todas as estruturas do ContextCF.jl;
- Estender a linguagem específica de domínio (DSL) inicialmente estabelecida pelo Persa e garantir seu funcionamento e extensão de forma semântica e previsível ao usuário;
- Disponibilizar alguns conjuntos de dados com contexto para uso imediato, facilitando uso do *framework* através de outro módulo complementar, *ContextDatasetsCF.jl*;
- Disponibilizar os módulos com código aberto para uso da comunidade científica.

1.2 Organização do Trabalho

Esse trabalho aborda o desenvolvimento do ecossistema para CARS e foi organizado da seguinte maneira:

- Capítulo 2: será abordada a fundamentação teórica necessária para o trabalho, fazendo uma introdução aos SR, abordando tópicos como filtragem colaborativa e *Context-Aware Recommender Systems* (CARS);
- Capítulo 3: serão abordados outros *frameworks* relevantes à proposta do *ContextCF.jl* e uma comparação entre os mesmos, evidenciando algumas diferenças

e decisões de design dos mesmos. Será abordada a arquitetura do *framework*, suas motivações e características;

- Capítulo 4: será abordado o uso dos módulos do *ecossistema*, mostrando alguns exemplos de uso do mesmo e decisões de projetos tomadas na construção dos mesmos;

- Capítulo 5: Conclusão e trabalhos futuros.

Capítulo 2

Sistemas de Recomendação

Os SR se tornaram parte integral da forma como exploramos grandes conjuntos de dados em sistemas de informações modernos, especialmente na web, como por exemplo: sugestões de amigos em redes sociais, ou sugestões de itens em *e-commerces*.

Quando o usuário sabe o que está procurando, ferramentas de pesquisa são capazes de recuperar informações que sejam úteis ao usuário, porém em muitos casos, o usuário não sabe o que está procurando e também não deseja gastar muito tempo vendo todo o conjunto de dados. Para resolver esse problema surgiram os SR, que geram recomendações de itens e/ou serviços relevantes ao usuário.

2.1 Introdução

SR são ferramentas de software e técnicas que fornecem sugestões de itens que sejam relevantes ao usuário. Recentemente eles têm se provado como meios valiosos de lidar com o problema da sobrecarga de informação, sugerindo ao usuário itens relevantes através das recomendações. Cada interação ou *feedback* do usuário sobre essas recomendações podem ser armazenados e usados posteriormente para gerar novas recomendações durante as próximas interações do usuário com o sistema (ADOMAVICIUS; TUZHILIN, 2015).

As sugestões tem relação com diversos processos de decisão, como quais itens

comprar, quais músicas ouvir ou o que ler. Com a grande quantidade de itens disponíveis nos dias de hoje ficaria inviável para o usuário avaliar todas as possibilidades e decidir o que consumir, o que torna os SR essenciais em muitos negócios, *e.g. e-commerces*. (KORTENHOF, 2017)

Empresas como *Google*[1], *Netflix*[2], *Spotify*[3] e *Amazon*[4] vêm utilizando os SR como forma de obter vantagens comerciais. Estudos demonstraram que os SR trazem três principais benefícios para os *e-commerces*: o aumento das vendas, vendas cruzadas e uma maior lealdade dos seus usuários. (KORTENHOF, 2017)

O problema de recomendação foi formalizado por Adomavicius e Tuzhilin (2005): seja U o conjunto de todos os usuários e I o conjunto de todos os itens. seja f a função utilidade que mede a importância do item i ao usuário u, *e.g.* $f : U \times I \rightarrow P$ onde $P \in R$ é o conjunto das preferências dos usuários por itens. Então, para cada usuário $u \in U$ é escolhido o item $i' \in I$ que maximize a função utilidade f. Ou seja:

$$\forall u \in U,\ i'_u = arg \max_{i \in I}\ f(u, i). \tag{2.1}$$

Cada elemento do espaço U pode ser definido como um perfil que inclua características do usuário *e.g.* idade, gênero, entre outras. Do mesmo modo, cada elemento do espaço I pode ser definido por um conjunto de características dos itens. Dentro de um contexto de *e-commerces*, as características podem ser: descrições textuais do produto, tipo de produto dentre outras.

A utilidade de um item em SR é representada normalmente por uma nota que indica a avaliação do usuário para um determinado item *e.g.* Maurício deu nota 4 de 5 para o filme Senhor dos Anéis. Entretanto, a função utilidade pode ser uma função arbitrária, definida pelo usuário ou calculada pelo sistema. Na tabela 2.1, vemos um fragmento de uma matriz $Usuário \times Item$ em um SR de filmes, onde o usuário avalia os filmes explicitamente.

[1]https://www.google.com
[2]https://www.netflix.com
[3]https://www.spotify.com
[4]https://www.amazon.com

	Poderoso Chefão	Matrix	Senhor dos Anéis
Paulo	2	ø	5
Filipe	3	5	ø
Maurício	4	5	4
Vitor	ø	1	4

Figura 2.1: Fragmento de uma matriz de notas de um Sistema de Recomendação de filmes

O problema central em SR é que a função utilidade f não é definida para todo o espaço $U \times I$, mas para um subconjunto dele, o que torna necessário extrapolá-la para todo o espaço. O objetivo de um Sistema de Recomendação é prever as notas de usuários para itens que ainda não tenham sido vistos e utilizar essas previsões para recomendação.

A extrapolação de f é passível de ser feita por heurísticas que definem a função utilidade e validam empiricamente sua desempenho ou estimando a função utilidade que otimiza certa métrica de desempenho *e.g.* erro quadrático médio (MSE), erro absoluto médio (MAE), dentre outras. Após estimar as notas dos itens não avaliados utilizando os métodos citados acima, o Sistema de Recomendação selecionará o item com nota de maior valor, ou os n itens melhores avaliados para o usuário.

A estimativa da nota de um item não avaliado por um usuário pode ser feita de diversas formas, seja através de técnicas de aprendizado de máquina, de teoria da aproximação ou através de várias outras heurísticas. A abordagem utilizada para estimar as notas é usada para classificar os algoritmos e desta forma as abordagens foram classificadas em cinco classes: (ADOMAVICIUS; TUZHILIN, 2015)

- **Baseada em conteúdo**: o SR recomendará itens semelhantes aos que o usuário gostou no passado;
- **Filtragem colaborativa**: a recomendação é feita baseado em itens que usuários de perfil semelhante gostaram no passado;
- **Demográfico**: a recomendação é feita baseada em nichos demográficos basea-

dos no perfil do usuário;

- **Baseada em conhecimento**: através de um conhecimento prévio de um domínio, são feitas inferências sobre as necessidades do usuário;
- **Abordagens híbridas**: combina quaisquer das abordagens cima.

Na próxima seção, será apresentada a filtragem colaborativa, a definição de contexto e o uso de informações contextuais na mesma.

2.2 Filtragem Colaborativa (CF)

As pessoas recorrem a opiniões de outras pessoas para obterem auxílio em tomadas de decisões sobre alguns itens desde os primórdios. Através dessas opiniões, uma pessoa poderia ponderar com relação à confiança dela pela fonte, ou até mesmo sob suas expectativas sobre o produto ou serviço. Esse tipo de recomendação é conhecida como recomendação boca a boca. A CF utiliza esse conceito como base e pode ser interpretada como um automatização desse tipo de recomendação.

Os usuários em um SR possuem diferentes perfis e objetivos no uso do mesmo. Para personalizar a experiência do usuário, os SR exploram diversas informações sobre os usuários. Os SR que usam como método filtragem colaborativa (CF) produzem suas recomendações para os usuários baseados em padrões de avaliações ou uso *e.g.* compras, sem a necessidade do uso de informações sobre os usuários ou itens sendo recomendados.

A CF possui diversos benefícios sobre a abordagem baseada em conteúdo, sendo o principal deles não depender de informação externa de usuários e itens, pois na mesma a qualidade da recomendação está diretamente relacionada à qualidade das informações disponíveis. Outro benefício é que a CF apresenta mais dinamicidade e diversificação quanto às recomendações para um usuário, visto que sua recomendação depende de um comportamento macro de usuários e itens, o que não ocorre na recomendação por conteúdo, cuja dependência de dados externos do usuário levam a super especialização. (ADOMAVICIUS; TUZHILIN, 2015)

Entretanto, existem diversos desafios nesta abordagem. As bases de um SR possuem uma grande quantidade de usuários e de itens, o que consequentemente torna a matriz $U \times I$ de um SR extremamente esparsa e o desempenho dos algoritmos está relacionado diretamente com a esparsidade da base (GIPSON, 2018). As bases de um SR possuem normalmente em torno de 1% de notas com relação ao total possível (POZZA et al., 2011a).

Outro problema decorrente da CF, é o surgimento de um novo usuário ou item, pois nesse tipo de abordagem é necessário um conjunto de avaliações para realizar uma predição, desta forma, se apenas as avaliações dos usuários são utilizadas para realizar a predição, a mesma falhará, já que não será possível comparar com nenhum outro usuário ou item no sistema, impossibilitando a recomendação desse item até o momento que ele obtiver alguma avaliação de algum usuário ou no caso do usuário, quando o mesmo avaliar algum item. Esse problema é conhecido como *cold start*. (ADOMAVICIUS; TUZHILIN, 2015)

Apesar dos desafios, a abordagem de CF obteve grandes sucessos na teoria e na prática (POZZA et al., 2011a), entretanto, existem ainda diversas questões a serem pesquisadas com o fim de superar os desafios intrínsecos à abordagem de maneira geral, como a esparsidade, escalabilidade dentre outros.

Na próxima seção, falaremos sobre a multidisciplinaridade do contexto e a relação entre as atividades de um usuário e seus contextos adjacentes.

2.3 Contexto

Contexto é um conceito multidisciplinar que tem sido pesquisado em diferentes campos de estudo, *e.g.* ciência da computação, ciência cognitiva, linguística, filosofia e psicologia. Desta forma, cada disciplina assume sua própria visão e abordagem, sendo mais específica que a definição genérica encontrada em dicionários "condições ou circunstâncias que afetam algo"(ADOMAVICIUS; TUZHILIN, 2015). Em SR, contexto é definido como qualquer informação útil para caracterizar a situação de uma entidade e que pode afetar como os usuários interagem com o sistema (ABOWD

et al., 1999). Um exemplo de contexto seria a utilidade da recomendação feita a um usuário u a um determinado filme i, usando como informação contextual ele estar acompanhado ou não, que poderia influenciar o filme que o mesmo irá consumir.

Devido à enorme variedade de definições entre diferentes disciplinas, inclusive em campos específicos dentro dessas disciplinas, em Pozza et al. (2011b), foram analisados mais de 150 definições de contexto em campos diferentes. Para ordenar essa diversidade, Dourish introduziu uma taxonomia a contextos, no qual os mesmos podem ser classificados em visões *representacionais* e *interacionais*. Na visão representacional, contexto é definido por um determinado conjunto de atributos observáveis pré definidos a estrutura (conhecido como *schema* em Bancos de Dados) que não se altera significantemente durante o tempo. Em outras palavras, assume que os atributos contextuais são identificáveis e conhecidos a priori, e, logo, podem ser capturados e usados em aplicações sensíveis a contexto. Em contraste, a visão interacional assume que o comportamento do usuário é induzido por contexto subjacente, mas que o contexto em si não é necessariamente observável. Além disso, Dourish assumiu que diferentes tipos de ações podem dar origem e demandar diferentes tipos de contextos relevantes, assumindo assim uma relação bidirecional entre atividades e contextos adjacentes: atributos contextuais influenciam atividades e diferentes atividades dão origem a diferentes contextos.

Na próxima seção falaremos sobre os *Context-Aware Recommender Systems* (CARS), o paradigma da obtenção de informações contextuais relevantes e seu uso na tentativa de adaptar contexto para o domínio específico dos SR.

2.4 Context-Aware Recommender Systems (CARS)

Modelar interesses dos usuários para atender às necessidades individuais é um dos grandes desafios em aplicações de filtragem de informação e recomendação (BELLOGÍN, 2012). É uma combinação do que está ocorrendo ao redor do usuário no momento da interação com o sistema (situação) e o que ele está tentando realizar (intenção) (WHITE; BAILEY; CHEN, 2009). Na abordagem utilizada pelos

Context-Aware Recommender Systems (CARS), a aplicação assume que a informação contextual é conhecida e definida, e que esses atributos contextuais afetam as avaliações, aumentando sua qualidade em relação aos SR que não a utilizam (ADOMAVICIUS; TUZHILIN, 2015). Neles, as avaliações são modeladas em uma função que não envolve apenas usuários e itens, mas também os atributos contextuais:

$$F : Usuario \times Item \times Contexto \rightarrow Predicão \tag{2.2}$$

Os CARS diferem dos *Context-driven Recommender Systems* (CDRS) no seu foco nas preferências do usuário ao invés do contexto. Enquanto os CDRS usam o contexto como sua principal fonte de informação para as recomendações, os CARS se adaptam às preferências do usuário, usando o contexto como informação adicional secundária para fazer a recomendação (WHITE; BAILEY; CHEN, 2009).

As variáveis de contexto podem ser obtidas de forma explícita, implícita ou dedutiva. Na forma explícita o sistema pergunta informações diretamente ao usuário, na implícita o sistema capta informações do usuário de forma mais sutil como dispositivo conectado, localização, horário de acesso, etc. Já na dedutiva são utilizados métodos estatísticos ou de mineração de dados para tentar descobrir a identidade de quem está utilizando determinado serviço, baseado no padrão de utilização do mesmo, *e.g.* ser um feriado, estar acompanhado ou não. (ADOMAVICIUS; TUZHILIN, 2015).

A informação contextual pode ser classificada como estática ou dinâmica (VILLEGAS, 2013). Quando estática, é assumido que essas informações são imutáveis durante o tempo, *e.g.* o aniversário de um usuário. Quando dinâmica, o contexto muda durante o tempo e afeta as necessidades do usuário, *e.g.* localização do usuário, tempo e temperatura (VILLEGAS; MÜLLER, 2010).

O uso da informação contextual no processo de recomendação leva à inserção de mais uma camada na recomendação. A forma que a obtenção de dados contextuais é modelada afeta no resultado da recomendação, assim como a forma de incorporar esse contexto nos SR (ADOMAVICIUS; TUZHILIN, 2015).

Na próxima subseção falaremos sobre a aquisição dos dados contextuais, sua

importância e as abordagens que podem ser usadas para tal.

2.4.1 Paradigmas para obtenção de dados contextuais

Os CARS existentes assumem a existência de certos fatores contextuais *e.g.* tempo, localização e clima para identificar o contexto em que as recomendações são oferecidas. Cada tipo de fator pode ser definido por sua estrutura, e os valores que as variáveis contextuais podem assumir e a sua forma de aquisição também podem variar dependendo de onde são obtidas (ADOMAVICIUS; TUZHILIN, 2015). A aquisição de dados contextuais, em qualquer das abordagens, deve ser levada como parte do processo de aquisição de dados. A decisão de quais variáveis contextuais devem ser relevantes e coletadas devem ser decididas no planejamento de um SR e durante o tempo de maneira iterativa. A relevância das variáreis contextuais varia dramaticamente de aplicação para aplicação *e.g.* localização para um *e-commerce* pode não ser tão relevante quanto para uma recomendação de restaurantes para um turista.

Muitos fatores contextuais podem não ser relevantes ou úteis para uma aplicação. Além da abordagem manual, onde as variáveis são sugeridas manualmente, existem também abordagens puramente computacionais, utilizando-se de técnicas de aprendizado de máquina, mineração de dados e estatística para decidir quais são relevantes ou não. Essas técnicas podem ser usadas em conjunto com a abordagem manual para melhores resultados (ADOMAVICIUS; TUZHILIN, 2015).

Na próxima seção, discutiremos as principais abordagens para incorporar contextos em um SR. Para isso, consideraremos que os dados relevantes já foram obtidos e armazenados nas bases de dados que serão usadas.

2.4.2 Paradigmas para incorporar o contexto representacional em SR

Após a obtenção de informação relevante de contexto usando a abordagem escolhida, a próxima etapa é a utilização desse contexto de maneira apropriada para produzir melhores recomendações. As abordagens para usar essa informação

contextual em SR podem ser generalizadas em dois grandes grupos: recomendação via pesquisa e consulta orientada a contexto, e recomendação via elicitação e estimação de preferências contextuais. A abordagem via pesquisa e consulta orientadas a contexto (*context-driven*) usa as informações contextuais para fazer a pesquisa em um certo conjunto de recursos e exibir os recursos que melhor correspondem a uma determinada pesquisa. A abordagem via elicitação e estimação de preferências contextuais *context-aware* utiliza as preferências do usuário e o contexto como parâmetros secundários para gerar a recomendação.

Existem três paradigmas para integrar informação contextual em CARS, dependendo da fase em que o contexto é inserido no processo de recomendação (VILLEGAS et al., 2018):

- **Pré-filtragem contextual**: a matriz de avaliações é filtrada antes do algoritmo de recomendação. Nessa filtragem, as interações na matriz que não são relevantes ao contexto atual são removidas, em seguida, as predições podem ser feitas usando qualquer algoritmo de recomendação tradicional;
- **Pós-filtragem contextual**: a informação contextual é ignorada inicialmente e as classificações são feitas aplicando-se algoritmos tradicionais de recomendação em todos os dados. O resultado é então filtrado de acordo com as informações que são relevantes ao usuário;
- **Modelagem contextual**: a informação contextual é diretamente integrada no modelo de recomendação.

Na próxima seção, discutiremos a Pré-filtragem e Pós-filtragem em detalhes, explicando suas vantagens e desvantagens.

2.4.3 Abordagens de contexto

De maneira geral, um SR tradicional pode ser descrito como uma função que utiliza dados de preferências parciais do usuário como entrada e produz uma lista de recomendações para cada usuário como saída. Após a função de recomendação ser

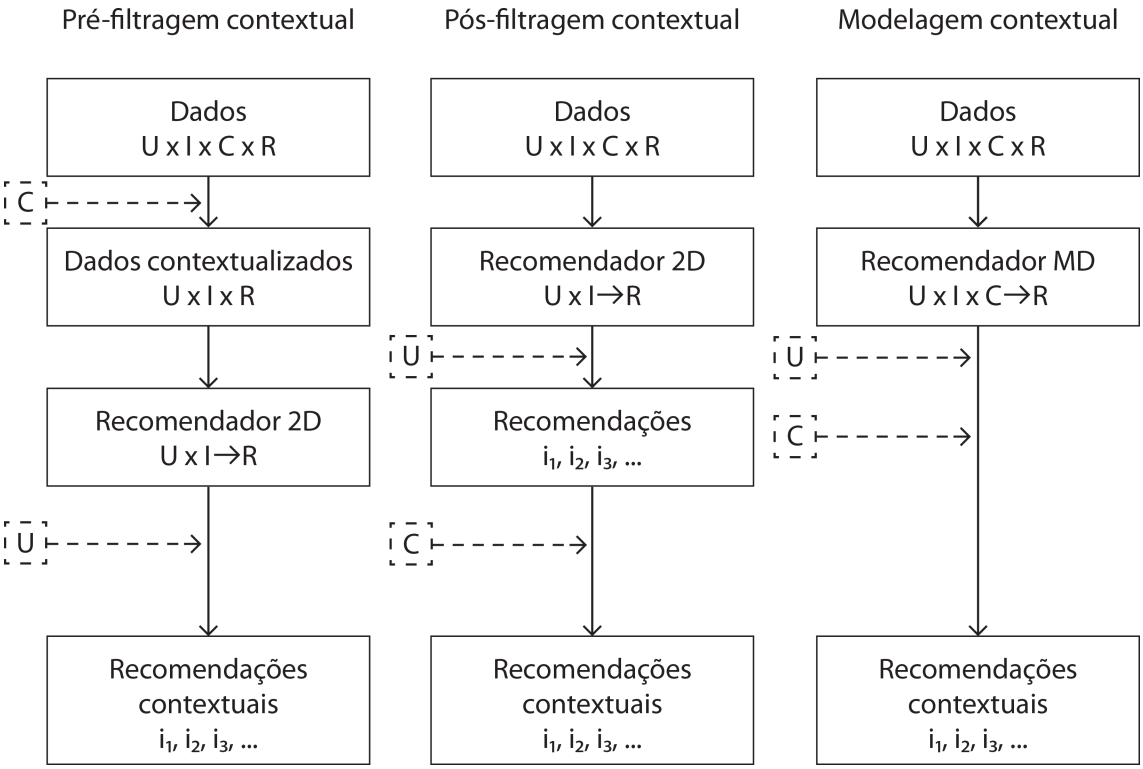

Figura 2.2: Abordagens de integração de contexto em CARS. **a)** Pré-filtragem contextual, **b)** Pós-Filtragem contextual, **c)** Modelagem contextual (VILLEGAS et al., 2018)

construída a partir dos dados disponíveis, a lista das recomendações é gerada usando a função de recomendação em qualquer usuário u e em todos os itens relevantes para obter a classificação de todos os itens e então se ordenar os itens de acordo com as notas obtidas. (ADOMAVICIUS; TUZHILIN, 2015).

O uso de informação contextual em cada um desses itens (função, item e usuário) gera diferentes abordagens para os CARS, conforme citado na seção 2.4.2.

2.4.3.1 Pré-filtragem contextual

Conforme mostrado na figura 2.2 (a), a pré-filtragem contextual usa os dados contextuais para construir a matriz usuário-item mais relevante para gerar recomendações, e portanto, permite a utilização de qualquer técnica de recomendação tradicional (ADOMAVICIUS; TUZHILIN, 2015).

Considerando a função descrita em 2.2, todo CARS utilizaria mais uma dimensão

em sua matriz: Usuário × Item × Contexto. Um caso de uso do contexto seria usar o clima em um determinado dia num recomendador de viagens. Nesse caso, a previsão do tempo seria utilizada para recomendar um trajeto ao usuário. Considerando W como o clima, teríamos que $U \times I \times W \rightarrow P$.

No exemplo acima a previsão do tempo poderia ser obtida de maneira implícita e estaria sempre disponível. Entretanto, isso não é aplicável a todos os casos de uso. Em casos mais específicos, as variáveis contextuais podem não ser facilmente obtidas ou estarem muito esparsas, o que prejudicaria a qualidade da recomendação.

Adomavicius e Tuzhilin (2015) propôs uma abordagem de *redução* do problema de recomendações contextuais multidimensionais para a abordagem tradicional de matriz de preferência Usuário-Item.

$$R^{D}_{Usuário \times Item \times Clima} : U \times I \times W \rightarrow predicão \tag{2.3}$$

onde U é o usuário, R é a função de recomendação, I é o item e W é o clima que será usado como contexto e D é o conjunto de dados que possui os registros (usuário, item, clima e avaliação) para cada usuário. Logo, a função de predição poderia ser expressada através da seguinte função de predição bidimensional:

$$\forall (i,u,w) \in U \times I \times W, R^{D}_{Usuário \times Item \times Clima}(u,i,w) = R^{D[Clima=w](Usuário,Item,Clima)}_{Usuário \times Item} \tag{2.4}$$

onde $D[Clima = w](Usuário, Item, Clima)$ representa um conjunto de classificações obtidas a partir de D através da seleção dos dados onde a dimensão $Clima$ tem valor w e mantendo os valores correspondentes a Usuário e Item além do valor da classificação em si. Esse exemplo representa um pré-filtro exato.

De forma resumida, podemos assumir que $D[Clima = w](Usuário, Item, Clima)$ é uma relação obtida a partir de D através da execução de duas operações relacionais: seleção seguida de projeção.

Essa abordagem permite que qualquer técnica de recomendação tradicional descrita na literatura possa ser usada. Entretanto, em alguns casos, a relação obtida

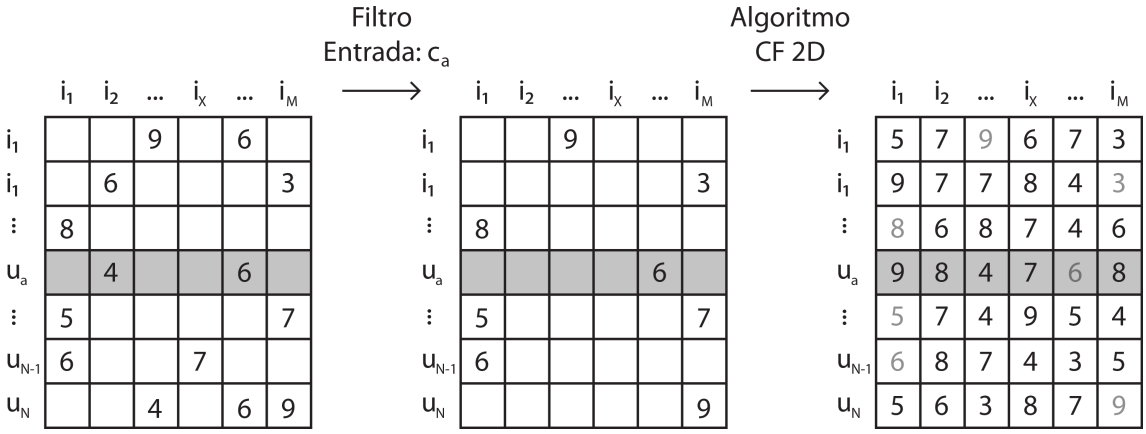

Figura 2.3: Pré-filtragem contextual (KORTENHOF, 2017)

pode não possuir classificações o suficiente para que o algoritmo de recomendação faça a predição (ADOMAVICIUS; TUZHILIN, 2015). Esse problema é conhecido como esparsidade na literatura de SR.

Adomavicius e Tuzhilin (2015) propôs uma abordagem mais genérica na redução multidimensional, onde não seria usado apenas o exato contexto w na predição (u, i, w), mas algum segmento de contexto S_w, que representa um superconjunto de um contexto w. Essa abordagem é conhecida como pré-filtragem generalizada.

Para avaliar o valor de uma sugestão de um restaurante R em um dia chuvoso para um usuário Pedro, $R^D_{Usuário \times Item \times Clima}(Pedro, R, Chuvoso)$, poderíamos expandir a seleção na relação para incluir dias nevando, por exemplo. Considerando então $S_w = D[w \in \text{“}chuvoso''|w \in \text{“}nevando'']$, a predição para cada para cada (u, i, w) seria:

$$R^D_{Usuário \times Item \times Clima}(u, i, w) = R^{D[Clima \in S_w](Usuário, Item, AGGR(Notas))}_{Usuário \times Item}(u, i) \qquad (2.5)$$

A abordagem de redução mostrada acima pode ser estendida de maneira genérica na redução de qualquer abordagem n dimensional. Na expressão acima, foi usada a notação $AGGR$ (nota), pois podem haver várias notas do mesmo usuário e item para diferentes Climas no conjunto de dados D, logo, é necessário agregar esses valores usando alguma função de agregação, *e.g.* média, ao reduzir a dimensionalidade do

espaço para recomendação. (ADOMAVICIUS; TUZHILIN, 2015)

2.4.3.2 Pós-filtragem contextual

Conforme mostrado na figura 2.2 (b), a pós-filtragem contextual ignora as informações contextuais ao gerar as recomendações. Então, ele ajusta a lista de recomendação para cada usuário usando as informações de contexto. Esses ajustes podem ser feitos *filtrando-se* as recomendações irrelevantes ou *ajustando* o valor das predições dado um determinado contexto.

Na pós-filtragem se busca, de maneira geral, analisar os dados de preferência de um usuário de forma a buscar padrões de uso dos itens pelos mesmos, gerando recomendações de melhor qualidade. Existem duas técnicas para pós filtragem: heurística ou baseada em modelo.

A abordagem heurística foca em encontrar características de itens comuns para um usuário em um determinado contexto, e então ajustam as recomendações filtrando os itens que não possuem um número significante dessas características, ou classificando os itens recomendados baseado no número de características relevantes que o item possui.

A abordagem baseada em modelo constrói modelos preditivos que calculam a probabilidade em que um usuário escolheria um certo tipo de item em um contexto e usa essa probabilidade para ajustar as recomendações, filtrando os itens que possuem probabilidade de ser relevante inferior a um certo limiar pré-definido ou classificando os itens ponderando as predições usando a probabilidade de ser relevante do mesmo dado um certo contexto. (ADOMAVICIUS; TUZHILIN, 2015)

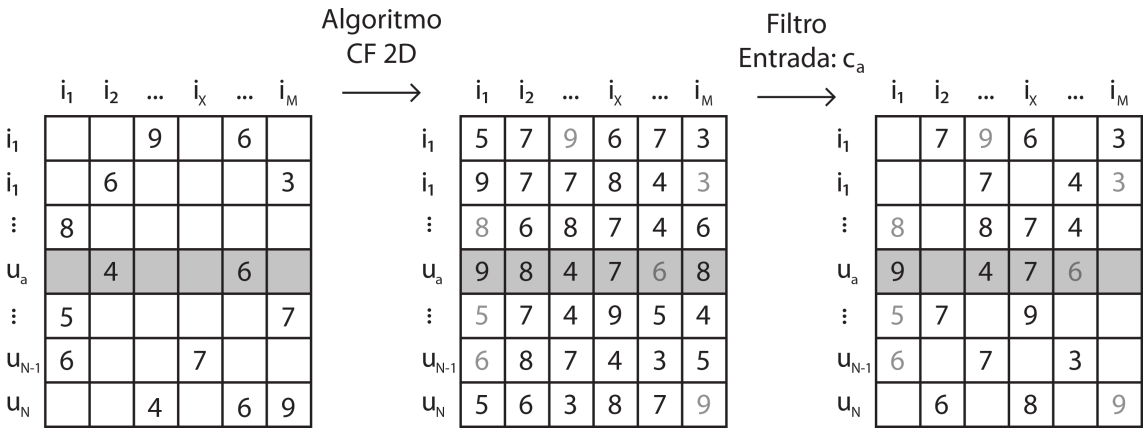

Figura 2.4: Pós-filtragem contextual (KORTENHOF, 2017)

Capítulo 3

ContextCF.jl

3.1 Motivação

A computação tem sido aplicada a uma infinidade de áreas, e devido a essa diversidade de uso, linguagens de programação com objetivos muito diferentes têm sido desenvolvidas. Dentre os domínios de aplicação temos como exemplo: as aplicações científicas, aplicações empresariais, inteligência artificial, programação de sistemas e desenvolvimento web.

Desde seu surgimento nos anos 1940, a computação digital tem sido utilizada para aplicações científicas. Aplicações científicas comumente possuem estruturas de dados relativamente simples, mas requerem diversas computações de aritmética de ponto flutuante. As primeiras linguagens de programação de alto nível para aplicações científicas foram projetadas para suprir tais necessidades, e tendo como competidora o *assembly*, performance era uma preocupação essencial (SEBESTA, 2009).

A programação científica é usada para aumentar a produtividade e reduzir o tempo de desenvolvimento de determinadas aplicações. Ela é aplicada na construção de modelos matemáticos e técnicas de soluções numéricas para analisar e resolver problemas em diversas áreas de domínio. Tradicionalmente, é separada em duas famílias de linguagem: as de produtividade e as de performance. Algumas características de linguagens produtivas como tipagem dinâmica e coletor de lixo

tornam o desenvolvimento exploratório e iterativo mais simples. Logo, é comum que diversas aplicações científicas comecem em uma linguagem de produtividade e posteriormente, com seu aumento de complexidade, sejam migradas para alguma linguagem de performance.

SR são atribuídos à computação científica e, dada as características acima, são considerados aplicações científicas, se beneficiando do uso de linguagens criadas com esse propósito. O valor das informações contextuais em SR foi demonstrada e suportada por diversos pesquisadores. A precisão das recomendações pode ser altamente afetada pelas informações de contexto, o que torna o seu estudo importante (VILLEGAS et al., 2018).

Desde o prêmio *Netflix*[1], osSR se tornaram um tema popular e atrativo, pois foi disponibilizado um conjunto de dados de larga escala com 100 milhões de avaliações de filmes, atraindo milhares de estudantes, engenheiros e entusiastas do campo. A natureza da competição incentivou o rápido desenvolvimento, onde os competidores construíram em cada geração de técnicas para melhorar a precisão da previsão. (ADOMAVICIUS; TUZHILIN, 2015)

Com isso, diversos *frameworks* começaram a surgir, com o intuito de facilitar o estudo, promovendo fácil acesso aos algoritmos clássicos para pesquisadores interessados no tema. Foram observadas algumas características comuns nos *frameworks*, como a possibilidade da parametrização em suas funções de predição; fácil extensão, facilitando a construção de novos algoritmos ou de novos conjuntos de dados; presença de métricas tradicionais do campo e em alguns casos, preocupação com desempenho e escalabilidade.

Na construção de um *framework*, diversas características de linguagens de produtividade são interessantes, pois algumas delas, como gerenciamento de memória são feitas pela linguagem, abstraindo esse problema do programador. Um grande exemplo de linguagem de produtividade seria o Python, que possui ampla aceitação na comunidade científica devido ao seu ecossistema científico *NumPy*[2], fornecendo

[1] https://www.netflix.com/
[2] http://www.numpy.org/

um *framework* de operações matemáticas e científicas que é estendido por um vasto número de módulos criados pela comunidade. Entretanto, Python[3] não é uma linguagem científica, e sim uma linguagem de propósito geral de alto nível, que foi estendida para atender às necessidades da comunidade científica. Isso leva ao problema das duas linguagens, onde os cientistas utilizam uma linguagem de alto nível, como R e Python, mas as partes críticas de desempenho precisam ser reescritas em C/C++, o que é extremamente ineficiente, apresentando erro humano e desperdiçando esforço.

Outra solução para aumento de produtividade utilizando linguagem de programação são as linguagens de domínio específico (DSL), criadas com o objetivo de solucionar problemas em um domínio específico de aplicações (FOWLER, 2010). Um exemplo seria uma abstração de uma DSL para uma matriz de avaliações, onde ao se especificar um usuário, item e seu contexto, é possível obter a nota do usuário para esse contexto, ao invés do dado na matriz original, simplificando a indexação dentro do domínio.

Dentre as vantagens em se utilizar uma DSL, temos o alto nível de abstração, eliminando diversos detalhes desnecessários de implementação, aumento na produtividade, permite que soluções sejam expressas no nível do domínio da aplicação e aumenta a legibilidade e concisão do código. Entretanto, seu uso traz também desvantagens, como o aumento nos níveis de indireção devido à adição de mais níveis de abstração, falta de suporte de ferramentas adequadas para desenvolvimento e manutenção da DSL, e curva de aprendizado de uma nova linguagem que possui aplicabilidade limitada ao domínio do problema (FOWLER, 2010).

Com o intuito de criar uma DSL que facilite o estudo de CARS para a comunidade científica, foi escolhido criar um módulo, estendendo o *framework Persa.jl*[4], por possuir uma estrutura robusta, facilmente extensível e que já usava um DSL própria, dando suporte ao mesmo, a variáveis contextuais e às operações básicas utilizadas por pesquisadores da área.

[3] https://www.python.org/
[4] https://github.com/JuliaRecsys/Persa.jl

3.2 Trabalhos Relacionados

Nessa seção destacaremos os *frameworks* de recomendação mais importantes e listaremos algumas de suas características que posteriormente serão analisadas dado alguns critérios de comparação.

3.2.1 Persa

Persa[5] é um *framework* criado para facilitar o estudo de filtragem colaborativa em Julia[6], focado em oferecer ferramentas que facilitem a construção de novos algoritmos, utilizando uma linguagem de computação científica, o que facilita o desenvolvimento. Ele estende a linguagem criando uma DSL e com isso aumentando a produtividade, pois o desenvolvedor não precisará se preocupar com acesso aos dados ou a construção das principais tarefas.

3.2.2 MyMediaLite

MyMediaLite[7] é uma biblioteca rápida e escalável de SR, focada em atender pesquisadores e estudantes. Ela aborda os dois cenários mais comuns da filtragem colaborativa: ranking e previsão, disponibilizando algoritmos para ambas, entretanto, não oferece suporte a informações contextuais (GANTNER et al., 2011).

3.2.3 CARSKit

CARSKit[8] é um motor de recomendações de código aberto, escrito em Java, especificamente projetado para CARS. Possui uma arquitetura flexível de maneira a ser fácil expandir o escopo dos algoritmos de recomendação contextuais e fornece espaço para o desenvolvimento de novos algoritmos no futuro. Oferece amplo suporte a informações contextuais (ZHENG, 2015).

[5] https://github.com/JuliaRecsys/Persa.jl
[6] https://julialang.org/
[7] http://www.mymedialite.net/
[8] https://github.com/irecsys/CARSKit

3.2.4 Surprise!

Surprise![9] é um módulo de expansão para o SciPy[10], escrito em Python para construção e análise de SR. Permite fácil extensão e escrita de novos algoritmos. Oferece uma parametrização extensiva e bem detalhada nos algoritmos de predição e documentação muito bem elaborada e com diversos exemplos de uso. Não oferece suporte a informações contextuais (HUG, 2017).

3.2.5 LightFM

LightFM[11] é uma implementação de diversos algoritmos de recomendação para feedback implícito e explicito. Permite incorporar metadados de usuário e item em algoritmos tradicionais de fatorização de matriz, representando cada usuário e item como a soma das representações latentes de suas características, assim permitindo generalizar recomendações para novos itens, e para novos usuários, através de suas características. É facilmente escalável para conjuntos de dados muito grandes e máquinas multicores através do uso de Cython[12], e é usado em produção por companhias como Lyst[13] e Catalant[14] (KULA, 2015).

3.2.6 RankSys

RankSys [15] é um *framework* para implementação e avaliação de técnicas e algoritmos de recomendação. Embora seja concebido como um *framework* para a experimentação genérica de tecnologias de recomendação, inclui apoio substancial com foco na avaliação e aprimoramento de novidade e diversidade. Foca explicitamente no problema de ranqueamento dos itens da recomendação, e não na predição como nos *frameworks* tradicionais. Não oferece suporte a informações contextuais.

[9] http://surpriselib.com/
[10] https://www.scipy.org/
[11] https://lyst.github.io/lightfm/docs/home.html
[12] https://cython.org/
[13] https://www.lyst.com/
[14] https://gocatalant.com/
[15] http://ranksys.org/

3.3 Comparação entre *Frameworks*

Foram analisados alguns *frameworks* de recomendação com o intuito de observar as características comuns e desejáveis para seu uso no estudo da área. O suporte a CARS foi limitado e dentre os seis analisados, apenas dois possuíam suporte para contexto, e nenhum deles usando linguagens de produtividade. Algumas características foram utilizadas para comparação entre os mesmos, conforme pode ser observado na tabela 3.1 sendo esses:

- Documentação: possuem documentação para uso, e em sua maioria com bons exemplos de uso do *framework*;
- Parametrização: Permitem a parametrização das funções de predição e afins, permitindo que o usuário consiga ajustar os algoritmos à sua necessidade;
- Extensibilidade: permitem usar um conjunto de dados do usuário e escrever suas próprias funções de predição usando a estrutura do *framework*, permitindo fácil extensão do *framework*;
- Modularidade: São separados em módulos, sendo cada módulo responsável por alguma interface do *framework e.g.* módulo de contexto, conteúdo, acurácia;
- Métricas: Oferecem algumas métricas para que o usuário possa comparar seus resultados e acurácia dos mesmos *e.g.* MAE, RMSE, MPE;
- Performance: Alguns *frameworks* oferecem parâmetros para otimização de desempenho dos algoritmos, como número de threads a serem utilizadas e otimizações em *multithread*;
- Linguagens de domínio específico: Alguns deles oferecem uma DSL para facilitar o uso e construção de algoritmos para o domínio.

Framework	Linguagem	Contexto	Param Predição	Documentação	Modularidade	Extensibilidade	Métricas	Otimizações	DSL
MyMediaLite *	C#	✓	✓	✓	✗	✓	✓	✗	✗
CARSKit	Java	✓	✓	✓	✓	✓	✓	✗	✗
Surprise!	Python	✗	✓	✓	✓	✓	✓	✗	✗
LightFM	Python	✗	✓	✓	✓	✓	✓	✓	✗
RankSys	Java 8	✗	✓	✓	✓	✓	✓	✓	✗
Persa	Julia	✗	✓	✗	✓	✓	✓	✓	✓

Tabela 3.1: Tabela de comparação de *frameworks* de recomendação

3.4 Linguagem de Programação

A programação científica tradicionalmente exige o mais alto desempenho, entretanto, grande parte dos programadores moveram para linguagens dinâmicas para o trabalho diário. Existem diversas razões para optar por linguagens dinâmicas *e.g.* tipagem dinâmica e gerenciamento de memória, e não é esperado que seu uso venha a diminuir. Novas técnicas de compilação e um design moderno na linguagem tornam possível eliminar a perda de desempenho e oferecer um ambiente único capaz de prototipar e eficiente o suficiente para implementar aplicações de alto desempenho (BEZANSON et al., 2018).

Na próxima subseção apresentaremos a linguagem Julia, mostrando seu histórico, seus objetivos como linguagem, algumas de suas características de implementação e decisões de design da linguagem.

3.4.1 Julia

A linguagem de programação Julia é uma linguagem dinâmica e flexível, apropriada para uso científico e numérico e com desempenho comparável a linguagens estaticamente tipadas (BEZANSON et al., 2018). Possuindo características como tipagem dinâmica, gerenciamento automático de memória e despacho múltiplo, características de linguagens de produtividade, permite também que os programadores possam gerenciar a memória. Além disso, usa um compilador *just-in-time* (JIT) e permite que o programador controle o layout da estrutura de dados na memória (BEZANSON et al., 2018).

O início do desenvolvimento de Julia se deu em 2009, com sua primeira versão sendo lançada em 2012. Desde seu lançamento, a comunidade aumentou, tendo mais

de 2 milhões de downloads em Agosto de 2018[16], atraindo alguns clientes de alto perfil e chegando a ser usada em 2015 pela *Federal Reserve Bank of New York* para fazer modelos da economia americana, onde foi observado um ganho de desempenho de dez vezes sobre a linguagem MATLAB[17].

Julia propõe aos programadores científicos a facilidade de uma linguagem de produtividade e o desempenho de uma linguagem de performance. Essa abordagem é promissora. Linguagens dinâmicas costumam sofrer perdas de desempenho de pelo menos uma ordem de magnitude ou mais com relação ao C (BEZANSON et al., 2018). Essa proposta solucionaria o problema das duas linguagens, pois a mesma poderia ser usada para desenvolvimento, facilitando o programador, e para aplicações que exigem desempenho, sem necessidade de reescrita de código.

3.4.1.1 Design da Linguagem

Inclui diversas funcionalidades comuns a muitas linguagens de produtividade, como tipagem dinâmica, tipagem opcional, reflexão e coletor de lixo. Uma de suas funcionalidades menos comum em linguagens de programação é conhecida como despacho múltiplo, ou seja, uma função pode possuir múltiplas implementações, distinguida pelos tipos nas assinaturas do método. Em tempo de execução, a chamada de função é despachada para o método mais específico aplicável para os tipos dos argumentos. No código 3.1, podemos ver a função *size* e suas múltiplas implementações.

```
julia> methods(size)
# 92 methods for generic function "size":
[1] size(B::BitArray{1}) in Base at bitarray.jl:70
[2] size(B::BitArray{1}, d) in Base at bitarray.jl:74
[3] size(r::Core.Compiler.StmtRange) in Base.IRShow
...
```

[16] https://juliacomputing.com/
[17] https://juliacomputing.com/case-studies/ny-fed.html

```
[92] size(f::IterTools.PeekIter) in IterTools
```

Código 3.1: Métodos possíveis para função *size*

As anotações de tipos em Julia também podem ser atribuídas a declarações de tipos de dados, onde, nesse caso, são verificadas sempre que os campos digitados são atribuídos. Julia diferencia também tipos concretos de abstratos, onde o primeiro pode ser instanciado e o ultimo estendido por subtipos.

3.4.1.2 Implementação da Linguagem

Segundo Bezanson et al. (2018) a implementação de Julia é mais simples que muitas linguagens dinâmicas. O compilador possui três otimizações que são executadas em uma representação intermediária de alto nível. A geração de código nativo é delegada para a infraestrutura do compilador LLVM[18]. As otimizações são: (1) *inlining* de métodos que desvirtualiza chamadas multi-despachadas e coloca *inline* o alvo da chamada; (2) *unboxing* de objetos para evitar a alocação na *heap*; e (3) especialização de métodos onde o código é especializado para seus tipos de argumentos reais.

Na figura 3.2, temos a declaração de um método genérico em Julia:

```
julia> function generic(a,b,c)
           return a + b + c
       end
generic (generic function with 1 method)
```

Código 3.2: Declaração de função genérica *generic*

Esse método, mesmo sendo genérico, possui mais de uma geração de código, dependendo dos tipos dos parâmetros passados na chamada da função. Isso permite a criação de métodos genéricos com uma grande otimização. Na figura 3.1 é possível ver a diferença entre o código gerado.

[18]https://llvm.org/

```
generic(1.0,2.0,3.0)

.text
  Function generic {
  Location: In[2]:2
  Function +; {
  Location: In[2]:2
          vaddsd  %xmm1,
          %xmm0, %xmm0
  }
 Function -; {
 Location: float.jl:397
          vsubsd  %xmm2,
          %xmm0, %xmm0
}
          retq
          nopl    (%rax)
}
```

```
generic(1,2,3)

.text
  Function generic {
  Location: In[2]:2
  Function +; {
  Location: In[2]:2
          leaq(%rdi,%rsi),
          %rax
  }
  Function -; {
  Location: int.jl:52
          subq    %rdx, %rax
}
          retq
          nopl    (%rax,%rax)
}
```

Figura 3.1: Código gerado por uma função genérica para parâmetros de entrada de diferentes tipos

3.5 Proposta

Conforme discutido anteriormente, o desempenho de SR que utilizam informações contextuais é superior aos que não utilizam (ADOMAVICIUS; TUZHILIN, 2015), logo, levar em consideração informações contextuais para fazer recomendações é imprescindível e conforme visto, não há *framework* para sistema de recomendação que atenda o uso de contexto, fácil de usar, facilmente extensível e que abstraia os conceitos de recomendação diretamente na linguagem. Nesse contexto, o *Persa* atende os requisitos acima possuindo uma DSL própria, alto grau de abstração, e devido a linguagem Julia (semântica e técnicas de linguagem de programação), consegue unir todas as características desejadas.

Esse trabalho então, se propõe a estender o *framework Persa*, seguindo os padrões de design de sua DSL e expandindo-a para suporte a informações contextuais. Essa extensão será feita de forma a atender as especialidades de recomendação com alto grau de abstração, dando suporte a contexto.

Na próxima subseção será levantada a modelagem do problema da recomendação,

o problema da recomendação com contexto e suas n-dimensões e os requisitos de contexto para criação de um *framework* com as características desejadas.

3.5.1 Modelagem

A abordagem de um SR de filtragem colaborativa pode ser representado pelo diagrama de classes conforme pode ser observado na imagem 3.2. As classes utilizadas nessa modelagem foram:

- *SistemaRecomendacao* representa o sistema de recomendação em si e armazena os usuários, itens e o conjunto de preferências;
- *ConjuntoPreferencia* representa o conjunto dos símbolos possíveis das preferencias no sistema *e.g.* de 1 a 5, de 1 a 10, gostei ou não gostei;
- *Preferencia* possível preferencia de um usuário para um item;
- *Usuario* representa o usuário no sistema, que efetua a avaliação no sistema e possui apenas o seu identificador único;
- *Item* representa o item/serviço no sistema, que pode ter sido avaliado ou não e possui seu identificador único e suas avaliações;
- *Avaliação* elemento que define que um usuário escolheu uma preferencia sobre o item.

Conforme pode ser observado na figura 3.2, o SR é representado pela interação entre duas grandes entidades: *usuários* e *itens*. No caso da filtragem colaborativa, a ideia é que através das preferências do usuário por itens, prever as suas avaliações. Dentro de um SR, existem opções de preferências pros usuários, *e.g.* gostei ou não gostei, números entre 0 a 5, que são representados na classe *Preferencia*. A classe *ConjuntoPreferencia* representa o conjunto de preferências possíveis no sistema, e a classe *Avaliação* define qual preferência um usuário escolheu para um item.

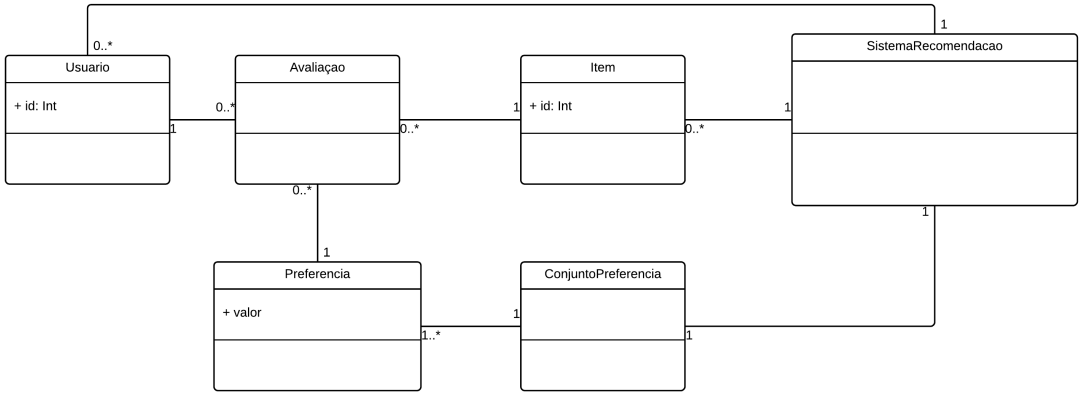

Figura 3.2: Diagrama de Classes de um Sistema de Recomendação

Essa modelagem é válida para o problema da recomendação, entretanto, a adição de informações contextuais ocasiona algumas alterações na modelagem, conforme pode ser observado no diagrama 3.3.

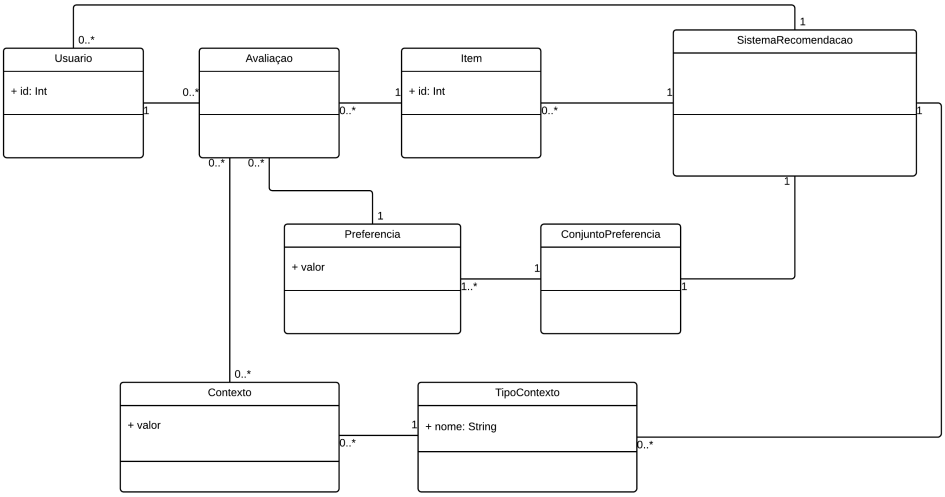

Figura 3.3: Diagrama de Classes de um CARS

Na modelagem de um CARS, a diferença é que uma avaliação pode possuir contextos, representados pela classe *Contexto*, e esses contextos estão agrupados em grandes tipos de contexto representandos pela classe *tipoContexto*, *e.g.* humor, clima. O SR também precisa possuir os tipos de contexto, para que eles possam ser utilizados nos contextos das avaliações feitas pelos usuários.

Na próxima subseção falaremos dos requisitos de fazer predições para CARS, as

principais abstrações do Persa e os requisitos para predição de notas com informação contextual.

3.5.2 Requisitos

O objetivo principal deste trabalho é construir uma estrutura em que seja fácil desenvolver e implementar os algoritmos de CARS, e possua as características já citadas anteriormente, de forma que se permita fazer as predições e o suporte seja feito diretamente na linguagem e não na biblioteca pelo uso de uma DSL própria. Considerando a tarefa básica de predição, é necessário elicitar os requisitos para predição de notas com informações contextuais.

O *Persa* considera que recomendação é o problema de navegar na matriz usuário-item, e quando se possui uma base dados, se armazena os dados logicamente e a partir disso utiliza-se um modelo para carregar essa base e efetuar o cálculo das notas. Ele possui duas grandes generalizações: a matriz com as avaliações e o modelo preditivo de notas.

Considerando uma variável chamada *Dataset* representando uma matriz de avaliações, podemos observar na imagem 3.4 a generalização na navegação na matriz de avaliação no *Persa* onde a linha representa um usuário u e a coluna um item i. Nela, existem três formas de se navegar na matriz: $Dataset[n]$, retorna a avaliação n não nula da base dados; $Dataset[u, i]$, onde é retornada a nota do usuário u para o item i e $Dataset[u, :]$ onde são retornadas todas as notas do usuário na base de dados.

O Persa possui uma segunda abstração importante em que ele interpreta o modelo preditivo como se fosse uma matriz usuário-item. Desta forma, o acesso aos dados e a previsão das avaliações se tornam os mesmos, assim uniformizando o acesso e considerando que a previsão também é uma matriz de avaliações, só que de *predições* de avaliações. Entretanto, essa matriz de predição, diferentemente da matriz usuário-item, não possuirá valores nulos pois o modelo gerará as predições das avaliações.

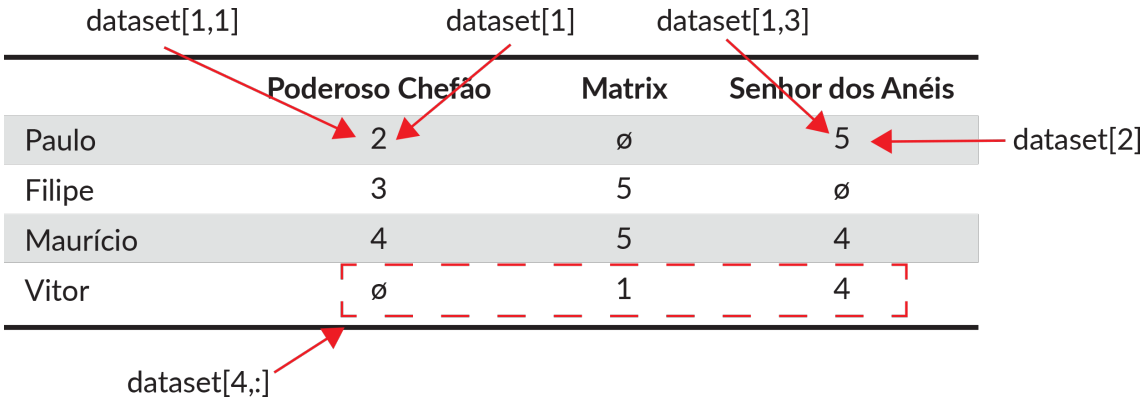

Figura 3.4: Navegação na matriz de avaliações no Persa

Considerando uma variável chamada *Model* representando um modelo de predição, e considerando que a navegação nessa matriz é semelhante a da matriz usuário item, teríamos: $Model[n]$, retornando a predição do item n pelo primeiro usuário na base de dados, $Model[u, i]$, retornando predição da nota do usuário u para item i e $Model[u, :]$ retornando as predições de avaliações do usuário u para todos os itens da base de dados, conforme podemos ver na imagem 3.5.

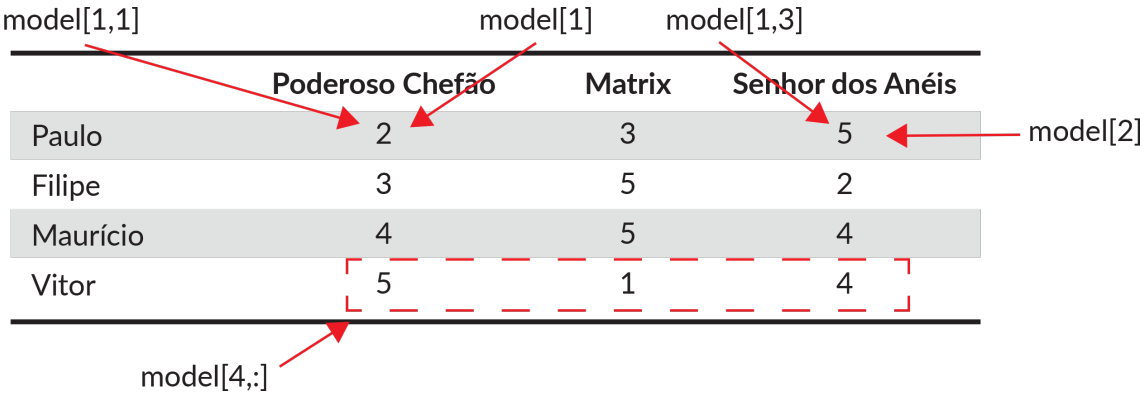

Figura 3.5: Navegação na matriz de predição $U \times I$ no Persa

Visando oferecer suporte a informações contextuais, a extensão do *Persa* foi feita de forma a respeitar as suas duas grandes abstrações, estendendo a navegação na matriz de avaliações, para a matriz usuário-item-contexto. O contexto pode ser acessado pela ordem numérica ou o símbolo correspondente, sendo o terceiro índice opcional.

No código 3.6, podemos ver a generalização da navegação na matriz de avaliações com contexto. Nela, além de ser possível acessar as avaliações do usuário através

das formas já estabelecidas pelo *Persa*, temos as formas contextuais, que adicionam um terceiro índice opcional: $dataset[n]$, retorna a avaliação n do do usuário u e as informações contextuais associadas; $dataset[u, i, c]$, retorna o valor de contexto c da avaliação do usuário u para o item i e $dataset[u, i, :]$, retornando todos os valores de contexto da avaliação do usuário u para o item i.

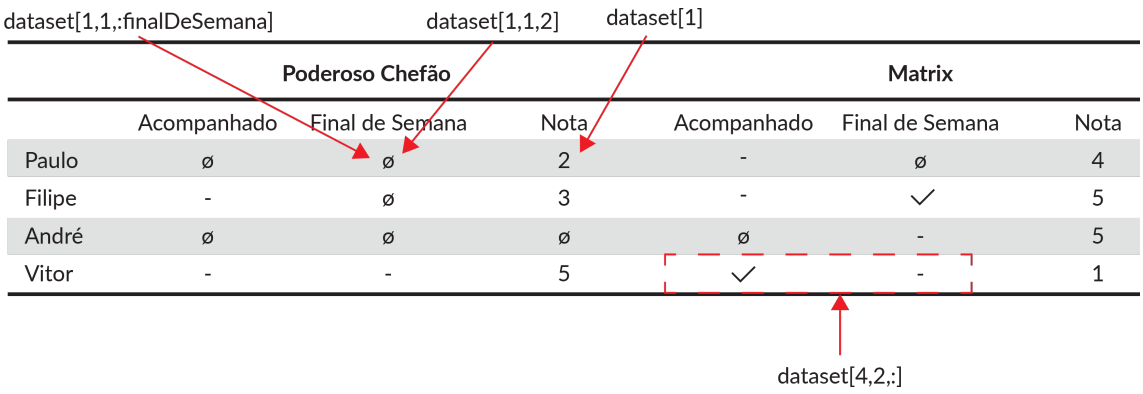

Figura 3.6: Navegação proposta na matriz de avaliações com contexto

Durante a elicitação de requisitos para extensão, foi concluído que não é necessário alterar o comportamento do modelo de predições do *Persa*, visto que essa interface será transparente ao usuário e será mantida.

Na próxima seção, veremos os detalhes de implementação para atendimento dos requisitos elicitados e alguns exemplos de uso.

Capítulo 4

Projeto

Visando facilitar o estudo de CARS e acelerar o aprendizado de pesquisadores e entusiastas da área, foi criado um ecossistema oferecendo dois módulos que abstraem dois problemas dos pesquisadores: um *framework* com estruturas facilmente extensíveis, suporte a contexto e fácil acesso a alguns *datasets* com informação contextual. O ecossistema criado consiste de dois módulos complementares ao *Persa*, sendo eles:

1. *ContextCF.jl* o módulo que estende o *framework Persa* e sua DSL, adicionando novas estruturas compatíveis e servindo como uma camada de abstração para uso de informação contextual;

2. *ContextDatasetsCF.jl*, módulo complementar que fornece alguns conjuntos de dados tradicionais de contexto para uso imediato do *framework ContextCF.jl*.

Nas próximas seções mostraremos os módulos que foram criados, como foram modelados, além de algumas decisões de projeto tomadas e alguns exemplos de uso dos mesmos.

4.1 ContextCF.jl

Com o objetivo definido de estender a abstração implementada pelo *Persa* para a matriz de avaliações com contexto, foi necessário utilizar as estruturas abstratas já

definidas pelo mesmo. Para os *ratings* foi utilizada a estrutura abstrata *AbstractRating* e para o *dataset* a estrutura abstrata *AbstractDataset*. Isso foi necessário pois os métodos do *Persa* usam esses tipos em suas assinaturas.

```
(v1.0) pkg> add ContextCF
```

Código 4.1: Adicionando ContextCF.jl para uso no Julia

No código 4.2, podemos ver as estruturas abstratas, onde T é o tipo genérico que deve ser do tipo *Number* (Int ou Float) e será usado para saber como retornar o tipo de avaliação.

```
abstract type AbstractDataset{T <: Number}
end

abstract type AbstractRating{T <: Number}
end
```

Código 4.2: Estruturas Abstratas do Persa

Conforme discutido previamente, a extensão do *Persa* foi feita através da adição dos dados de meta contexto no *Dataset* e dos valores de contexto nas preferências. Nas próxima subseções, veremos em detalhes as extensões feitas nas estruturas e as funções utilitárias que foram sobrecarregadas ou adicionadas para atender os requisitos que foram elicitados.

4.1.1 Estruturas

Na abstração da matriz usuário-item feita pelo *Persa*, o mesmo utiliza duas estruturas primárias que foram estendidas para dar suporte a dados contextuais: *Dataset* e *Ratings*. Nas próximas subseções veremos as estruturas originais do *Persa* e como foram feitas suas extensões.

4.1.1.1 Dataset

O suporte à abstração da matriz de avaliações é um dos requisitos básicos elicitados, e sua extensão nos possibilita navegar na mesma com alto nível de abstração. No código 4.3 podemos ver a estrutura *Dataset* no *Persa*, composta de uma matriz esparsa de *ratings*, o conjunto de preferências, o número de usuários e número de itens.

```
struct Dataset{T <: Number} <: AbstractDataset{T}
    ratings::SparseMatrixCSC{AbstractRating{T}, Int}
    preference::Preference{T}
    users::Int
    items::Int
end
```

Código 4.3: Estrutura *Dataset* no Persa

No código 4.4, podemos ver a inserção de uma nova estrutura, *metaContext*, que armazena as tuplas $nomeColunaContexto => Tipo$. Essa informação é necessária pois se tratando de informação contextual, não é possível prever o tipo do dado e para determinadas funcionalidades, ter esse dado disponível é relevante aos usuários do módulo.

```
struct DatasetContext{T <: Number} <: Persa.AbstractDataset{T}
    ratings::SparseMatrixCSC{Persa.AbstractRating{T}, Int}
    preference::Persa.Preference{T}
    users::Int
    items::Int
    metaContext::Dict{Symbol,Type}
end
```

Código 4.4: Estrutura *DatasetContext* no módulo ContextCF

Essas adições permitem a extensão da DSL do *Persa* conforme proposto na subseção 3.5.2, e permite que as abstrações exibidas no código 3.6 sejam válidas. Em 4.5, podemos ver alguns exemplos de navegação na matriz de avaliações com contexto

na base de dados *MovieLens 100k*, representado pela variável *dataset*, usando os módulos do ecossistema.

```
julia> dataset[1]
(user: 1, item: 1, rating: ContextCF.ContextRating{Int64}
(5, Dict{Symbol,Any}(:timestamp=>874965758)))

julia> dataset[1,1]
ContextCF.ContextRating{Int64}
(5, Dict{Symbol,Any}(:timestamp=>874965758))

julia> dataset[1,1,1]
874965758

julia> dataset[1,1,:timestamp]
874965758
```

Código 4.5: Extensão da DSL do Persa pelo módulo ContextCF para *DatasetContext*

4.1.1.2 Ratings

O suporte ao uso de informações contextuais nos *ratings* permite seu fácil uso e manipulação pelos algoritmos de predição. É possível verificar no código 4.6 a estrutura *Rating* no *Persa*, composta do valor da avaliação e um objeto de preferência, que armazena e valida os possíveis valores que a nota poderia assumir no *Dataset*.

Podemos observar durante a construção do objeto o uso da função *correct*, responsável por corrigir o valor das avaliações antes de sua inserção para o valor mais próximo no conjunto de preferências *e.g.* em um *dataset* com conjunto de preferência entre 1 e 5, uma avaliação de valor 4.2 seria corrigida para nota 4.

```
struct Rating{T <: Number} <: AbstractRating{T}
    value::T
    Rating(x::T, preference::Preference{T})
    where T <: Number = new{T}(correct(x, preference))
end
```

Código 4.6: Estrutura *Rating* no Persa

No código 4.7 podemos observar a adição de uma nova estrutura, *context*, que armazena as tuplas $nomeColunaContexto => ValorContexto$ referentes a cada *rating*. As informações contextuais são esparsas, conforme dito anteriormente, logo, para as informações de contexto não presentes no *dataset*, foi usado o objeto *singleton missing* disponibilizado pelo Julia, visto que a ausência de certos dados de contexto também pode ser usado para melhorar predições em determinados casos.

```
struct ContextRating{T <: Number} <: Persa.AbstractRating{T}
    value::T
    context::Dict{Symbol,Any}
    ContextRating(x::T, preference::Persa.Preference{T})
    where T <: Number =
    new{T}(Persa.correct(x, preference), context)
end
```

Código 4.7: Estrutura *ContextRating* no ContextCF

Essa adição permite a extensão da DSL do *Persa* para *ratings* conforme proposto na subseção 3.5.2, e permite que as abstrações de *ratings* sejam válidas. No código 4.8, podemos ver um exemplo de uso da DSL usando um *ContextRating* na base de dados *MovieLens 100k*, usando os módulos do ecossistema.

```
julia> using ContextCF
julia> using ContextDatasetsCF

julia> dataset = ContextDatasetsCF.MovieLens()

julia> rating = dataset[1,1]
ContextCF.ContextRating{Int64}
(5, Dict{Symbol,Any}(:timestamp=>874965758))

julia> rating[:timestamp]
874965758

julia> rating[1]
874965758
```

Código 4.8: Extensão da DSL do Persa pelo módulo ContextCF para *ContextRating*

4.1.2 Funções Auxiliares

Visando facilitar o desenvolvimento de novos algoritmos, algumas funções do *Persa* foram sobrecarregadas para dar suporte às informações contextuais e outras foram criadas para o funcionamento do módulo conforme necessário. Nelas, temos as funções auxiliares básicas utilizadas em estruturas de controle e repetição, essenciais no desenvolvimento de qualquer algoritmo. Em seguida, veremos alguns exemplos do uso dessas funções, considerando a variável *noContextDataset* como um conjunto de dados sem informação contextual e a variável *contextDataset* como um conjunto de dados com informação contextual.

Conforme pode ser observado no código 4.9, a função *size* foi reescrita para retornar como terceiro parâmetro o número de tipos de informações contextuais no *dataset*. Essa função oferece uma maneira rápida de obter as métricas de qualquer *dataset*.

```
julia> size(noContextDataset)
(943, 1682)

julia> size(contextDataset)
(943, 1682, 1)
```

Código 4.9: Sobrecarga da função *size* pelo módulo ContextCF

Pode ser observado no código 4.10 a função utilitária *context*, que pode ser usada tanto no *DatasetContext*, quanto no *ContextRating*, retornando os dados sobre o contexto nos mesmos e podendo ser usado para iterações.

```
julia> ContextCF.context(contextDataset)
Base.KeySet for a Dict{Symbol,Type} with 1 entry. Keys:
  :timestamp

julia> ContextCF.context(contextRating)
Base.KeySet for a Dict{Symbol,Any} with 1 entry. Keys:
  :timestamp
```

Código 4.10: Função *context* adicionada pelo módulo ContextCF

No código 4.11 é possível observar a função utilitária *value*, que permite a obtenção do valor do *rating* diretamente, sem acessar diretamente na sua estrutura. Essa função foi sobrecarregada para retornar o valor também em um *ContextRating* e estendida para retornar o valor da informação contextual a partir do símbolo que representa a mesma.

```
julia> value(rating)
5

julia> value(contextRating)
5

julia> value(contextRating,:timestamp)
874965758
```

Código 4.11: Sobrecarga da função *value* pelo módulo ContextCF

No código 4.12 pode ser observado a sobrecarga na função *getindex*, no módulo Base, que permite a sobrecarga de funções padrão em um tipo, ou de maneira geral, como indexação, operações matemáticas dentre outras. Essa sobrecarga possibilita a navegação na matriz $u \times i \times c$ através do índice numérico opcional *e.g.* $matriz[1, 1, 1]$. Esse tipo de sobrecarga foi feita em todas as operações exibidas na figura 3.6.

```
function Base.getindex(
  dataset::DatasetContext,
  user::Int, item::Int,
  contextColumn::Int
  )

  if contextColumn > length(dataset.metaContext)
    throw(Error("This context column doesn't exist."))
  end

  values(dataset.ratings[user,item].context)[contextColumn]
end
```

Código 4.12: Sobrecarga da função *getindex* pelo módulo ContextCF

4.2 ContextDatasetsCF.jl

Com o intuito de facilitar o uso e oferecer alguns conjuntos de dados de filtragem colaborativa com informação contextual para experimentação imediata, foi criado um módulo que, utilizando as estruturas do ContextCF.jl, efetua o download desses conjuntos de dados automaticamente, mapeia os dados e retorna um objeto válido *DatasetContext* para estudo de CARS.

```
(v1.0) pkg> add ContextDatasetsCF
```

Código 4.13: Adicionando ContextDatasetsCF.jl para uso no Julia

4.2.1 Conjuntos de Dados

Nessa seção falaremos sobre alguns dos conjuntos de dados disponibilizados pelo módulo, além de exemplos de uso dos mesmos.

4.2.1.1 MovieLens 100k

O conjunto de dados MovieLens 100k (MILLER et al., 2003) é oriundo de um sistema de recomendação de filmes desenvolvido pelo GroupLens[1] e possui 100.000 avaliações. Contém 943 usuários e 1682 filmes, com conjunto de preferência variando de 1 a 5.

Além das avaliações, existem informações como idade e o sexo, e sobre os filmes, como título e data de lançamento. Essas informações não foram utilizadas pois descaraterizariam a proposta dentro do contexto de CARS em filtragem colaborativa. Como forma de informação contextual, foi utilizado o *timestamp* da avaliação.

```
julia> ContextDatasetsCF.MovieLens()
```

Código 4.14: Exemplo de uso do conjunto de dados Movielens

[1] https://grouplens.org/

4.2.1.2 MovieLens 1M

O conjunto de dados MovieLens 1M (HARPER; KONSTAN, 2015) é oriundo de um sistema de recomendação de filmes desenvolvido pelo GroupLens[2] e possui 1.000.209 avaliações. Contém 6040 usuários e 3706 filmes, com conjunto de preferência variando de 1 a 5.

Assim como no conjunto de dados na subseção 4.2.1.1, além das avaliações existem informações como idade e o sexo, e sobre os filmes, como título e data de lançamento. Essas informações não foram utilizadas pois descaraterizariam a proposta dentro do contexto de filtragem colaborativa. Como forma de informação contextual, foi utilizado o *timestamp* da avaliação

```
julia> ContextDatasetsCF.MovieLens1m()
```

Código 4.15: Exemplo de uso do conjunto de dados Movielens 1m

4.2.1.3 InCarMusic

O conjunto de dados InCarMusic (BALTRUNAS et al., 2011) é oriundo de uma aplicação mobile que oferece recomendações aos passageiros do carro após a inserção de algumas avaliações pelos usuários em uma aplicação web e possui 4012 avaliações. Contém 42 usuários e 139 itens, com conjunto de preferência variando de 1 a 5.

Assim como no conjunto de dados na subseção 4.2.1.1, além das avaliações existem informações como idade e o sexo, e sobre os filmes, como título e data de lançamento. Essas informações não foram utilizadas pois descaraterizariam a proposta dentro do contexto de filtragem colaborativa. Como forma de informação contextual, foram usados os seguintes dados, conforme exibido na tabela 4.1.

```
julia> ContextDatasetsCF.InCarMusic()
```

Código 4.16: Exemplo de uso do conjunto de dados InCarMusic

[2] https://grouplens.org/

Fator Contextual	Condições Contextuais
Estilo de Condução	Descontraída, esportiva
Tipo de Estrada	Cidade, rodovia, serpentina
Tipo de Paisagem	Costa, interior, montanhas/colinas, urbano
Sonolência	Acordado, Sonolento
Condições do Tráfego	Estrada livre, muitos carros, engarrafamento
Humor	Agitado, feliz, preguiçoso, triste
Clima	Nublado, nevando, ensolarado, chuvoso
Momento do dia	Dia, manhã, tarde, noite

Tabela 4.1: Informações Contextuais no conjunto de dados InCarMusic

4.2.1.4 TripAdvisor

O conjunto de dados TripAdvisor[3], foi extraído das avaliações online do site. Possui apenas um tipo de contexto: tipo de viagem (Família, Casal, Negócio, Sozinho e amigos). É composto de 14175 avaliações, 2731 usuários e 2269 hotéis (ZHENG; MOBASHER; BURKE, 2014).

Além das avaliações, dados sobre os usuários e os hotéis também estão disponíveis, mas não foram utilizados pois descaracterizariam a proposta.

```
julia> ContextDatasetsCF.TripAdvisorV2()
```

Código 4.17: Exemplo de uso do conjunto de dados TripAdvisor

[3]https://www.tripadvisor.com.br/

Capítulo 5

Conclusões

5.1 Considerações finais

Foi desenvolvido um ecossistema para CARS, estendendo o *framework Persa* com o objetivo de facilitar seu estudo e abstrair alguns detalhes do programador, facilitando seu trabalho e aumentando sua produtividade.

Para tornar a extensão válida, a DSL já implementada pelo *Persa* foi estendida e informações contextuais foram adicionadas, de forma a facilitar seu uso no desenvolvimento de novos algoritmos de predição para CARS.

O uso de Julia trouxe ao módulo suas características de performance e produtividade, além de fornecer o arcabouço necessário para criação de uma DSL com suporte diretamente na linguagem, possuindo os requisitos necessários para extensão.

Os módulos desenvolvidos para o ecossistema de CARS são de código aberto e estão disponíveis no repositório oficial de pacotes de Julia, fornecendo acesso rápido e facilitado para qualquer entusiasta ou profissional da área.

5.2 Limitações e trabalhos futuros

Durante o desenvolvimento do módulo, algumas decisões de projetos foram tomadas de maneira empírica, pois a usabilidade de um *framework* remete diretamente à forma que os usuários farão seu uso. O uso extensivo do mesmo por outros usuários, especialmente especialistas na área e seu *feedback* são essenciais para o desenvolvimento contínuo do mesmo e aceitação, fornecendo os recursos e abstrações desejáveis para o domínio, aumentando a adesão e aumento da base de usuários e contribuidores.

O desenvolvimento futuro de algumas das operações mais usadas por pesquisadores em CARS, como o *splitting*, e sua adição futura no *framework* certamente teria muito valor aos usuários do mesmo, assim como uma documentação com maior número de exemplos, implementação de algoritmos de predição que utilizem as informações contextuais, implementação de métricas específicas para contexto para que se possa avaliar os resultados obtidos por esses mesmos algoritmos, além de comparação entre resultados de algoritmos tradicionais usando informações contextuais através do uso de técnicas como pré-filtragem e pós-filtragem.

Referências

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. 1999.

ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. v. 17, n. 6, 2005. ISSN 10414347.

ADOMAVICIUS, G.; TUZHILIN, A. Context-aware recommender systems. *Recommender Systems Handbook, Second Edition*, n. September, 2015. ISSN 0738-4602.

BALTRUNAS, L. et al. Incarmusic: Context-aware music recommendations in a car. In: *E-Commerce and Web Technologies*. [S.l.]: Springer, 2011.

BELLOGÍN, A. Recommender System Performance Evaluation and Prediction: An Information Retrieval Perspective. n. October, 2012.

BEZANSON, J. et al. Julia : Dynamism and Performance Reconciled by Design. v. 1, 2018.

DOURISH, P. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, v. 8, n. 1, Feb 2004. ISSN 1617-4917.

FOWLER, M. *Domain Specific Languages*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321712943, 9780321712943.

GANTNER, Z. et al. MyMediaLite. *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*, 2011.

GIPSON, A. House Bill 950, Mississippi Legislature 2018 Regular Session. *Mississippi Legislature*, n. February, p. 2, 2018. ISSN 00224790 (ISSN).

HARPER, F. M.; KONSTAN, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, ACM, New York, NY, USA, v. 5, n. 4, dez. 2015. ISSN 2160-6455.

HUG, N. *Surprise, a Python library for recommender systems*. 2017.

KORTENHOF, B. L. V. Context-Aware Recommender Systems in the E-commerce Domain. 2017.

KULA, M. Metadata embeddings for user and item cold-start recommendations. In: BOGERS, T.; KOOLEN, M. (Ed.). *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.* [S.l.]: CEUR-WS.org, 2015. (CEUR Workshop Proceedings, v. 1448).

MILLER, B. N. et al. Movielens unplugged: Experiences with an occasionally connected recommender system. In: *Proceedings of the 8th International Conference on Intelligent User Interfaces*. New York, NY, USA: ACM, 2003. (IUI '03). ISBN 1-58113-586-6.

POZZA, A. et al. Magnetic Resonance Enterography for Crohn's Disease: What the Surgeon Can Take Home. *Journal of Gastrointestinal Surgery*, v. 15, n. 10, p. 1689–1698, 2011. ISSN 1091255X.

POZZA, A. et al. Magnetic Resonance Enterography for Crohn's Disease: What the Surgeon Can Take Home. *Journal of Gastrointestinal Surgery*, v. 15, n. 10, 2011. ISSN 1091255X.

SEBESTA, R. *Conceitos de Linguagens de Programação - 9.ed.* [S.l.]: Grupo A - Bookman, 2009. ISBN 9788577808625.

VILLEGAS, N. M. Context management and self-adaptivity for situation-aware smart software systems. *University of Victoria*, 2013.

VILLEGAS, N. M.; MÜLLER, H. A. Managing dynamic context to optimize smart interactions and services. In: _____. *The Smart Internet: Current Research and Future Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 289–318. ISBN 978-3-642-16599-3.

VILLEGAS, N. M. et al. Characterizing context-aware recommender systems: A systematic literature review. *Knowledge-Based Systems*, Elsevier, v. 140, 2018.

WHITE, R. W.; BAILEY, P.; CHEN, L. Predicting user interests from contextual information. 2009. ISSN 09254773.

ZHENG, Y. A User's Guide to CARSKit. 2015. ISSN 1548-825X.

ZHENG, Y.; MOBASHER, B.; BURKE, R. Contexts recommendation using multi-label classification. In: *Proceedings of the 13th IEEE/WIC/ACM International Conference on Web Intelligence (WI 2014)*. [S.l.: s.n.], 2014.