

API REST para Sistemas de Recomendação: Projeto e Implementação

Hans Buss Heidemann

Departamento de Informática e Estatística Universidade Federal de Santa Catarina (UFSC) Florianópolis, Brasil hans.buss@grad.ufsc.br

Roberto Willrich

Departamento de Informática e Estatística Universidade Federal de Santa Catarina (UFSC) Florianópolis, Brasil roberto.willrich@ufsc.br

Resumo—text

O presente trabalho especifica uma API REST e desenvolve uma arquitetura de implementação de um Web Service de recomendação conforme o padrão da API-REST. Através destas API, qualquer sistema Web autenticado pode se registrar, atualizar o perfil dos usuários, e solicitar recomendações para um dado usuário. A solução utiliza o framework Connexion para API e Surprise para certos algoritmos de recomendação (SVD, CoClustering, KNNBasic) e Scikit-Learn para outros (baseado em conteúdo). Foram realizados testes com o dataset MovieLens-1m que apresentaram 77.7% de precisão e 27.5% de recall em comparação com o Surprise localmente (77.6% e 27.5%) e validaram a viabilidade de realizar o desacoplamento da recomendação via Web Services REST.

Index Terms—sistemas de recomendação, API REST, web services, perfil do usuário, desacoplamento

I. INTRODUÇÃO

Cada vez mais os usuários da Web têm a sua disposição uma enorme quantidade de recursos digitais (como músicas, filmes e documentos em geral), além de serviços e produtos. Neste artigo, o termo *item* será utilizado para referenciar qualquer um destes recursos, serviços e produtos. Devido ao montante de *itens* disponíveis para os usuários da Web, a sobrecarga de informação é um problema recorrente, quando a tarefa do usuário em localizar *itens* relevantes se torna complicada, devido ao grande número de *itens* recuperados nas buscas na Web [1].

Sistemas de Recomendação (SR) podem ser utilizados para solucionar este problema de sobrecarga de *itens* [2], tendo como principal objetivo auxiliar no processo de identificação de *itens* relevantes aos interesses e preferências dos usuários [3]. Tais sistemas auxiliam o usuário no processo de localização de *itens* que sejam relevantes com base de técnicas de filtragem de informação, removendo informações que sejam irrelevantes ou desnecessárias à determinada pessoa.

Na maior parte dos sistemas Web que se beneficiam de SR, a implementação dos serviços de recomendação ocorre de forma integrada (ou fortemente acoplada) aos demais módulos do sistema e ao seu banco de dados. À medida que a complexidade dessa implementação cresce, suas limitações começam a aparecer, como, por exemplo, a alta dependência entre componentes do sistema, o alto custo necessário para a manutenção, além da pouca flexibilidade e reusabilidade, em

que até mesmo uma pequena mudança pode afetar diversos componentes do sistema, limitando a adaptação do mesmo para uso em outros contextos [4].

A utilização de uma implementação baseada em Web Services (WS) [5] permite que o SR seja desenvolvido de forma independente do sistema Web que fará uso das recomendações, resultando em diversos benefícios, como a possibilidade do SR ser utilizado por outros sistemas Web, aumento na resiliência da implementação, a possibilidade da utilizar sistemas operacionais, linguagens e ferramentas diversas e o desacoplamento do serviço-cliente.

Este artigo apresenta o projeto e implementação de uma API REST (*Application Programming Interface - Representational State Transfer*) para SR, permitindo o desacoplamento entre algoritmos de recomendação e aplicações clientes. Está organizado como segue: a Seção II apresenta a fundamentação teórica sobre WS, REST e SR; a Seção III descreve a API e arquitetura propostas; a Seção IV detalha a implementação do protótipo; a Seção V apresenta a avaliação experimental; e a Seção VI conclui o trabalho com discussão de trabalhos futuros.

Existem propostas na literatura que envolvem a implementação de WS para a utilização de SR [6]. No entanto, essas propostas se baseiam na implementação de um algoritmo de recomendação para WS e não na utilização de um framework já existente para a implementação de algoritmos adicionais. [7]

II. FUNDAMENTAÇÃO TEÓRICA

A. Web Services e REST

1) *Web Services*: Como definido pelo W3C *Web Services Architecture Working Group*, um WS é uma aplicação de software identificada por uma URI (*Uniform Resource Identifier*), nas quais as interfaces são capazes de serem definidas, descritas e descobertas como artefatos XML [8]. Tais aplicações permitem a interação direta com outros sistemas Web utilizando mensagens baseadas em XML e/ou JSON, trocadas via protocolos de Internet, normalmente utilizando HTTP/HTTPS, mas não restrito a tais, podendo serem utilizados outros mecanismos como o FTP.

As vantagens na utilização de WS incluem a desacoplagem de interface do serviço da implementação, a vinculação

dinâmica de serviços, além do aumento da interoperabilidade multilinguagem e multi-plataforma.

2) *Representational State Transfer (REST)*: Para [9], o REST é um estilo arquitetural para sistemas de hipermeia distribuídos criado para guiar o projeto e o desenvolvimento da arquitetura da Web. Esse estilo ignora os detalhes dos componentes da implementação e da sintaxe do protocolo ao propor um conjunto de restrições de como a arquitetura de sistemas de hipermeia distribuídos, como a Web, devem funcionar.

O estilo REST [9] define 6 restrições relacionadas a estilos arquiteturais que garantem escalabilidade, simplicidade e independência de componentes: Cliente-Servidor (separação entre interface e armazenamento), Protocolo sem estado, *Cache*, Interface Uniforme, Sistema em camadas e *Code-On-Demand*, que é opcional.

B. Sistemas de Recomendação

Sistemas de recomendação (SR) estão sendo cada vez mais utilizados na Web, seja no momento da busca por um filme ou um *item* em um site de *e-commerce*, ou então na hora de receber um *post* de um conhecido em uma rede social. SRs estão presentes nesses momentos, com o objetivo apoiar a localização de *itens* que sejam de interesse do usuário em meio a quantidade gigantesca de *itens* disponíveis na Web. Desta forma, os SRs auxiliam em minimizar o problema chamado de *Sobrecarga de itens* [1].

Existem várias definições para SRs, que variam dependendo do autor que se consulta. Para [10], um SR tem por objetivo apoiar o processo social de indicar ou receber recomendações, no qual se busca a identificação um conjunto de *itens* com base nos interesses do usuário. Outros autores, como [11], definem SR como um tipo específico de sistemas de filtragem de informação, utilizados para a identificação de um conjunto de *itens* relevantes ao usuário.

1) *Filtragem Colaborativa (FC)*: A FC é uma técnica de filtragem que pressupõe que existe uma grande probabilidade de que um usuário goste dos *itens* que atendem as preferências de outros usuários com perfis similares ao usuário foco da recomendação [1], [12]. Desta forma, a FC busca identificar grupos de usuários com perfis similares, os chamados de Vizinhos Próximos, e, a partir destes, realizar a recomendação com base na estimativa de avaliações (*ratings*) que o usuário foco da recomendação poderia atribuir aos *itens*.

2) *Filtragem Baseada em Conteúdo (FBC)*: Na FBC, a recomendação de *itens* é realizada a partir da comparação entre o perfil do usuário foco da recomendação e as características associadas ao *item*, como metadados ou seu próprio conteúdo. Para isso, a FBC observa o comportamento do usuário para a partir disso recomendar *itens* que tenham relação com o perfil do usuário, onde são definidas as preferências e interesses do usuário com base nas características dos *itens* que o mesmo gostou, podendo terem sido avaliados ou não. [1], [13].

C. Frameworks de Recomendação

1) *Critério de Seleção*: Os critérios escolhidos para o levantamento dos frameworks de recomendação foram os seguintes:

- Possuir código aberto;
- Ser implementado utilizando a linguagem Python em versão maior ou igual a Python3;
- Suportar o uso do algoritmo de recomendação considerando *datasets* customizados para geração da recomendação;
- Ser de fácil instalação no Sistema Linux.

Tabela I
COMPARAÇÃO DE Frameworks DE RECOMENDAÇÃO

Framework	Foco	Algoritmo	heightSurprise	Implementação
Predição <i>QRec</i>	Implementação	Pred./Rank. ^a	<i>LightFM</i>	Implementação
Predição <i>Cornac</i>	Comparação	<i>Ranking</i>	<i>Case Rec.</i>	Construção
Predição <i>LensKit</i>	Avaliação		Predição <i>Elliot</i>	Experimentos
Predição <i>height</i>				

^a Pred. = Predição; Rank. = Ranking

2) *Comparaçao e Escolha*: O framework *Surprise* [14] foi escolhido após análise da Tabela I, considerando os seguintes critérios: (i) documentação completa e bem estruturada; (ii) suporte nativo a *datasets* customizados via *Pandas DataFrame* e CSV; (iii) 11 algoritmos consolidados na literatura (SVD, SVD++, NMF, KNN, etc.); e (iv) medidas de similaridade integradas (coseno, Pearson). Embora *QRec* ofereça mais algoritmos (30+), o *Surprise* apresenta melhor equilíbrio entre funcionalidade, facilidade de uso e maturidade da biblioteca para o escopo deste trabalho.

III. Web Service DE RECOMENDAÇÃO PROPOSTO

A. Visão Geral

A utilização do WS de recomendação proposto por sistemas Web ocorre de forma desacoplada através de uma API REST. Para a utilização é necessário ser criado um contexto, onde serão mantidas as informações acerca dos usuários, *itens* e algoritmos de recomendação configurados.

São oferecidos pela API métodos para: (i) criação e gerenciamento de contextos; (ii) registro das ações dos usuários; (iii) registro das características dos *itens*; e (iv) solicitação de recomendações.

B. Conceitos Básicos

Contexto: Cada sistema Web cliente do WS deve criar um contexto. Um contexto mantém todas as informações necessárias para a realização de recomendações aos usuários de um dado contexto, considerando os *itens* oferecidos por este contexto.

Ações: São representações das interações entre usuários e *itens* utilizadas pelos algoritmos de recomendação. Podem ser avaliações (*rating*), curtidas (*likes*) ou compras (*buy*) e conter parâmetros associados ou não.

Autenticação: Na presente proposta, o método de autenticação adotado é a utilização de um *token* de acesso.

Neste caso, será retornada uma chave de acesso no campo *X-Auth-Token* do cabeçalho após o contexto ter sido registrado no WS sendo necessário que este *token* seja enviado no cabeçalho das requisições no campo *X-Auth-Token* para ser autenticado no sistema.

Métricas: Existem diferentes unidades disponíveis no WS, sendo utilizadas para definir valores quantitativos relacionados à ação. Por exemplo, nas avaliações de *itens*, o contexto pode definir métricas como 5-estrelas, *likes*.

Itens: Caso o cliente desejar utilizar algoritmos de recomendação que considerem características dos *itens* a recomendar, ele deve realizar o registro das características de todos os *itens* disponíveis no cliente e que sejam passíveis de recomendação. Os seguintes tipos de *itens* são aceitos pela API: (i) *Item genérico*; (ii) Vídeo; (iii) Música; e (iv) Livro.

Algoritmos: O WS de recomendação proposto oferece a seus clientes um conjunto expansível de algoritmos de recomendação referenciado por um identificador único e com uma série de parâmetros customizáveis. Os algoritmos disponíveis são: recomendações baseadas em popularidade, previsão de *rating* (*SVD*, *CoClustering* e *KNNBasic*) e baseada em conteúdo. Cada algoritmo pode ser referenciado por um identificador único, e possui uma série de parâmetros que podem ser customizados pelos clientes. Na criação do contexto, o cliente deverá informar a lista de algoritmos adotados no contexto com suas respectivas parametrizações.

C. Métodos da API

A Tabela II resume os principais métodos oferecidos:

Tabela II
PRINCIPAIS MÉTODOS DA API REST

Método	Endpoint	Função
POST	/api/context	Criar contexto
DELETE	/api/context/{id}	Remover contexto
POST	/api/action	Registrar ações
POST	/api/item	Registrar itens
GET	/api/recommendation	Obter recomendações

D. Arquitetura do WS

A arquitetura proposta é composta pelos seguintes módulos, ilustrados na figura 1: **Gerenciador de Contexto:** Responsável pela criação, configuração e remoção de contextos. Também é responsável pela autenticação do cliente; **Atualizador de Contextos:** Responsável por armazenar os dados das ações e *itens* utilizados para recomendação; **Gerenciador de Recomendação:** Responsável por gerenciar a entrega de recomendações para o cliente; **Atualizador de Recomendações:** Responsável por gerar e armazenar novas recomendações periodicamente; **Adaptador de Modelos:** Responsável por adaptar os dados para a utilização nos SR e por adaptar as saídas do SR para o cliente;

O fluxo de operação é composto de quatro etapas: (i) o cliente autentica-se e recebe o *token* de acesso pelo Gerenciador de Contexto; (ii) o Atualizador de contextos persiste as ações de usuários e características dos *itens*; (iii) periodicamente

são geradas listas pré-calculadas de recomendação através do Atualizador de Recomendações; e (iv) o Gerenciador de Recomendação atende a requisições de recomendações consultando as bases já computadas.

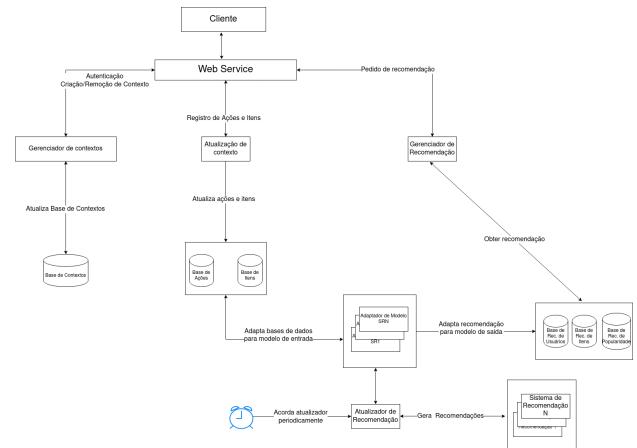


Figura 1. Arquitetura do *Web Service* de Recomendação

IV. IMPLEMENTAÇÃO

A. Tecnologias Utilizadas

Com o objetivo de selecionar o *framework Web* que seria utilizado, foi realizado uma análise entre os principais *frameworks* disponíveis (Django [15], Flask [16], Falcon [17], Connexion [18], Fastapi [19] e Hug [20]). A partir dela foram selecionados o *Connexion* para implementação da API REST e o *Surprise* [14] para a utilização dos algoritmos de recomendação.

O *Connexion* foi selecionado por proporcionar uma estratégia *API-First*, permitindo que a lógica da API seja primeiro descrita utilizando padrões da *OpenAPI* e depois mapeada para funções que envolvem as técnicas de recomendação, sem necessidade de mexer na implementação da API. Além disso, por ter sido construído sobre o *framework Flask*, é possível estender suas funcionalidades facilmente utilizando o ecossistema de *plugins* existentes.

Em relação aos algoritmos de recomendação, o *framework Surprise* foi selecionado, devido a sua documentação completa, a facilidade de operar com *datasets* personalizados e possuir diversos algoritmos implementados consolidados na literatura. Além de possuir métricas de avaliação integradas e a capacidade de lidar nativamente com formato CSV e *Pandas DataFrames*.

B. Protótipo Desenvolvido

1) *API Implementada:* No módulo *Web Service* foram implementados os métodos essenciais para o funcionamento da API. Estão disponíveis métodos para gerenciar contextos (*createContext* e *deleteContext*), para registrar novas ações e *itens* (*registerAction* e *registerItem*) e para receber uma nova recomendação personalizada (*getRecommendation*)

2) *Sistemas de Recomendação*: Foram implementados cinco algoritmos de recomendação, sendo um deles baseado em popularidade, três baseados no *rating* dos *itens* e o último baseado no conteúdo dos *itens*.

O algoritmo baseado em popularidade foi implementado utilizando o número de *ratings* de cada *item*, cada *item* tinha o seu número de *ratings* somado e aqueles que possuem o maior número são considerados os mais populares e são recomendados.

Aqueles baseados em *ratings* foram implementados utilizando os algoritmos do *framework Surprise* e são eles: *SVD* [21], *CoClustering* e *KNN* (*K-Nearest Neighbors*). Todos seguem a mesma estrutura, que envolve gerar o conjunto de treinamento a partir do *dataset*, treinar os algoritmos com o conjunto de treinamento e após isso é gerado o conjunto de testes para obter as previsões. As previsões são então mapeadas para cada usuário e retornadas as mais altas.

Para a implementação do algoritmo baseado em conteúdo foi utilizado a biblioteca de *Machine Learning Scikit-Learn* [22] e envolveu utilizar um dos campos do *item*, como o título. Primeiro se verifica a importância das palavras presentes no campo utilizando o algoritmo *TF-IDF* [23] (*Term frequency-inverse document frequency*) e se remove palavras consideradas inúteis, como "the". Após obter a frequência de cada palavra, é calculado a similaridade do cosseno para cada *item*, recomendando aqueles que possuem o menor ângulo e ignorando os que possuem um ângulo muito aberto.

3) *Cliente de Teste*: Para avaliar o WS implementado, um protótipo de cliente foi desenvolvido utilizando o *dataset ML-1m*, disponibilizado através do *framework Surprise*.

O desenvolvimento do protótipo envolveu a criação de um contexto no WS e a configuração dos algoritmos de recomendação, tempo de atualização das recomendações e a métrica a ser utilizada. Após isso, foi realizado o registro das ações presentes no *dataset* utilizado, contendo informações do usuário, do *item*, o valor associado a métrica e a *timestamp* da ações.

V. AVALIAÇÃO EXPERIMENTAL

A. Metodologia de Avaliação

Para validar o correto funcionamento da proposta, foi realizada uma rodada de testes que envolvia receber recomendações de filmes do *MovieLens-1m* [24] a partir de um sistema cliente que fazia o uso da API e de uma instância do *Surprise* rodando localmente.

A realização dos testes ocorreu uma técnica de validação cruzada conhecida como *K-fold* [25], embaralhando o *dataset* de forma aleatória e separando o mesmo em *k* grupos, sendo *k* = 5. Após a divisão do *dataset*, foi rodado algoritmo de recomendação *SVD* [21] em cada um dessas subdivisões para a geração das previsões. As métricas de precisão e *recall* foram calculadas considerando relevantes os *itens* com *rating* maior ou igual 4.0 na escala de 1 a 5 do *MovieLens-1m*.

Tendo acesso as previsões foram calculadas as métricas desempenho, *precisão* e *recall* apresentadas nas tabelas III e IV. A primeira indica, dentro do total de *itens* recomendados,

quants são realmente relevantes, enquanto a segunda indica quantos dos *itens* considerados relevantes foram recomendados.

B. Resultados

Tabela III
COMPARAÇÃO DE PRECISÃO

Iteração	Surprise	Web Service	height1
0.7754	0.7786	2	0.7801
0.7750	0.7801	3	0.7786
0.7747	0.7815	5	0.7748
0.7778 height			

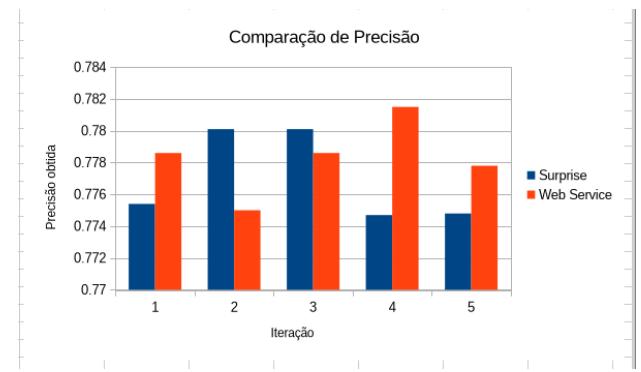


Figura 2. Comparação da métrica de Precisão entre implementação local (*Surprise*) e *Web Service* proposto utilizando algoritmo *SVD* no *dataset MovieLens-1m* com validação cruzada 5-fold.

Tabela IV
COMPARAÇÃO DE RECALL

Iteração	Surprise	Web Service	height1
0.2746	0.2760	2	0.2763
0.2765	0.2771	3	0.2745
0.2759	0.2750	5	0.2737
0.2775 height			

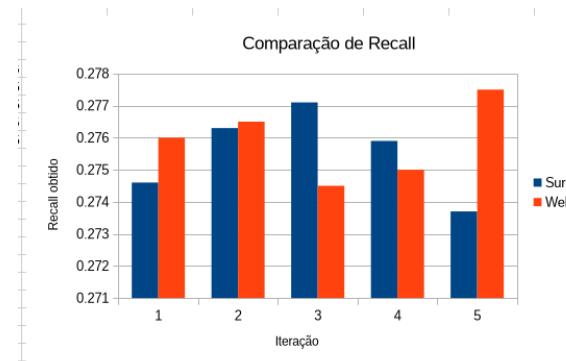


Figura 3. Comparação da métrica de *Recall* entre implementação local (*Surprise*) e *Web Service* proposto utilizando algoritmo *SVD* no *dataset MovieLens-1m* com validação cruzada 5-fold.

C. Discussão

As Figuras 2 e 3 ilustram visualmente a comparação entre as implementações ao longo das cinco iterações. Analisando as medidas obtidas, é possível perceber que os resultados obtidos pelo cliente e pelo *Web Service* diferem entre si. Analisando os dados registrados no WS, foi possível perceber que houve uma perda de dados quando foi simulado o envio do *dataset MovieLens-1m* [24] para o WS. A perda de dados ocorreu devido a limitação do computador onde a simulação estava acontecendo, onde o cliente e o servidor operavam em uma única máquina.

No entanto, mesmo com essas limitações é possível realizar observações sobre as medidas obtidas. Os dados de precisão média obtidos pelo cliente local (77.60%) apresentaram uma pequena diferença na taxa de acertos em relação ao WS (77.83%), significando que o WS indica mais *itens* que são relevantes do que comparado com o cliente local. Já quando analisamos a taxa de *recall*, ambos os sistemas possuem uma taxa praticamente idêntica (27.55% e 27.54%), indicando desempenho similar na recomendação de *itens* considerados relevantes. O *recall* obtido reflete a capacidade SVD [21] de identificar todos os *itens* potencialmente relevantes quando se lida com um *data* esparsos como o *MovieLens-1m* [24] que possui aproximadamente 95% das entradas vazias na *matrix usuário-item*: Recomenda apenas *itens* com alta confiança de relevância, resultando em *recall* moderado com precisão elevada, priorizando qualidade sobre quantidade nas recomendações.

VI. CONCLUSÃO

Este trabalho desenvolveu uma API REST para desacoplar SRs de aplicações cliente, utilizando *Connexion* para a implementação da API e o *Surprise* para os algoritmos de recomendação escolhidos através de uma revisão do estado da arte.

Foram utilizados quatro componentes principais para implementar a arquitetura proposta: Gerenciador de Contexto, responsável por realizar a autenticação e criação de contextos; Atualizador de Contexto, responsável por armazenar ações de usuários e características de *itens*; Gerenciador de Recomendação, responsável por entregar as recomendações e o Adaptador de Modelos, que permite integrar modelos de *frameworks* diferentes sem modificar a API.

Foram implementados cinco algoritmos: recomendação por popularidade, três algoritmos de predição (SVD, *CoClustering* e *KNNBasic*) e um baseado em conteúdo utilizando TF-IDF. A validação da implementação foi feita utilizando o *dataset MovieLens-1m* utilizando validação cruzada com 5 partições, obtendo precisão média de 77.7% e *recall* de 27.5%, valores semelhantes aos do *Surprise* (77.6% e 27.5%). Embora a precisão seja satisfatória, o *recall* baixo revela uma limitação: apenas cerca de um quarto dos *itens* relevantes são efetivamente recomendados, refletindo a esparsidade natural dos dados quanto limitações do algoritmo para explorar todo o espaço de *itens*, além da sobrecarga computacional observada no sistema cliente durante testes.

O trabalho demonstrou a viabilidade técnica de desacoplar SR através de *Web Services REST*, contribuindo com uma API documentada que pode servir de referência para implementações futuras e um mecanismo de adaptação que facilita a integração de diferentes *frameworks* de recomendação.

Este estudo apresenta algumas limitações que devem ser consideradas: (i) testes realizados em ambiente monomáquina; (ii) apenas o algoritmo SVD foi avaliado experimentalmente; e (iii) a perda de 8% dos dados durante transmissão.

Para trabalhos futuros é recomendado: (i) desacoplamento dos algoritmos de recomendação em servidores dedicados; (ii) adição de novos algoritmos e *frameworks* (*QRec*, *Cornac*, *LightFM*) ao *Web Service*; e (iii) realização de testes de desempenhos em cenários com múltiplos contextos simultâneos e utilizando *datasets* com volumes de dados acima dos utilizados (como *MovieLens-25M*).

REFERÊNCIAS

- [1] A. Salles, “Serviço web de recomendação baseado em ontologias e grafos para repositórios digitais,” 2017.
- [2] A. Chernev, U. Böckenholz, and J. Goodman, “Choice overload: A conceptual review and meta-analysis,” 2015.
- [3] S. C. Cazella, M. A. S. N. Nunes, and E. B. Reategui, “A ciência da opinião: Estado da arte em sistemas de recomendação,” 2010.
- [4] P. Kavanagh, Oct 2018. [Online]. Available: <https://www.linkedin.com/pulse/three-disadvantages-tightly-coupled-connections-pavinee-kavanagh>
- [5] E. Costa and J. Aguiar, Janderson e Magalhães, “Sistemas de recomendação de recursos educacionais: conceitos, técnicas e aplicações,” 2013.
- [6] E. Aïmeur, D. Daboué, F. S. M. Onana, D. Benslimane, and Z. Maamar, “Wsrs: A web service recommender system,” in *International Conference on Web Information Systems and Technologies*, 2007. [Online]. Available: <https://api.semanticscholar.org/CorpusID:5987914>
- [7] P. Pandharbale, S. N. Mohanty, and A. K. Jagadev, “Study of recent web service recommendation methods,” in *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, 2020, pp. 692–695.
- [8] C. Ferris and J. Farrell, “What are web services?” *Commun. ACM*, vol. 46, no. 6, p. 31, jun 2003. [Online]. Available: <https://doi.org/10.1145/777313.777335>
- [9] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [10] P. Resnick and H. R. Varian, “Recommender systems,” *Commun. ACM*, vol. 40, no. 3, p. 56–58, mar 1997. [Online]. Available: <https://doi.org/10.1145/245108.245121>
- [11] T. Ruotsalo, “Methods and applications for ontology-based recommender systems,” 01 2010.
- [12] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Trans. Inf. Syst.*, vol. 22, no. 1, p. 5–53, Jan. 2004. [Online]. Available: <https://doi.org/10.1145/963770.963772>
- [13] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” in *The adaptive web: methods and strategies of web personalization*, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin, Heidelberg: Springer, 2007, pp. 325–341.
- [14] N. Hug, “Surprise: A python library for recommender systems,” *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020. [Online]. Available: <https://doi.org/10.21105/joss.02174>
- [15] Django Software Foundation, “Django.” [Online]. Available: [https://django.org](https://.djangoproject.com)
- [16] M. Grinberg, *Flask web development: developing web applications with python*. “O'Reilly Media, Inc.”, 2018.
- [17] “GitHub - falconry/falcon: The no-magic web data plane API and microservices framework for Python developers, with a focus on reliability, correctness, and performance at scale. — github.com,” <https://github.com/falconry/falcon>, [Accessed 07-10-2023].

- [18] J. Santos, H. Jacobs, R. Carfcio, R. Sneyders, J. Finkhaeuser, D. Grossmann-Kavanagh, D. Dutra, Ruwann, G. Wilson, J. Kleijn, C. Clauss, L. Apple, E. Zanko, L. B., C. Guillemot, D. DeFisher, N. Musatti, M. Valkonen, P. Horacek, Ibigpapa, V. Ruiz, D. Hotham, F. Bruggisser, fp dsem, J. Bryan, T. Puschkasch, dfeinzeig, K. Indyk, J. Boecquaert, and J. Black, “spec-first/connexion,” 10 2023. [Online]. Available: <https://github.com/spec-first/connexion>
- [19] “GitHub - tiangolo/fastapi: FastAPI framework, high performance, easy to learn, fast to code, ready for production — github.com,” <https://github.com/tiangolo/fastapi>, [Accessed 07-10-2023].
- [20] “GitHub - hugapi/hug: Embrace the APIs of the future. Hug aims to make developing APIs as simple as possible, but no simpler. — github.com,” <https://github.com/hugapi/hug>, [Accessed 07-10-2023].
- [21] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in python,” *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2825–2830, Nov. 2011.
- [23] G. Salton and C. Buckley, “Term weighting approaches in automatic text retrieval,” USA, Tech. Rep., 1987.
- [24] F. M. Harper and J. A. Konstan, “The movielens datasets: History and context,” *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.
- [25] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.

INFORMAÇÕES SOBRE O TRABALHO ORIGINAL

Link para Dissertação Original

Este artigo é baseado no Trabalho de Conclusão de Curso:

Título: Projeto e Implementação de uma API REST para Sistemas de Recomendação

Autor: Hans Buss Heidemann

Instituição: Universidade Federal de Santa Catarina (UFSC)

Ano: 2023

Link: <https://repositorio.ufsc.br/handle/123456789/253264?show=full>

Alterações Realizadas

A principal alteração em relação ao trabalho original foi a criação de dois novos gráficos (Figuras 2 e 3) apresentando uma visualização gráfica comparativa entre os resultados obtidos pelo *framework Surprise* e pelo *Web Service* implementado.

Esta figura complementa as Tabelas III e IV presentes no trabalho original, facilitando a visualização e interpretação dos resultados experimentais. O gráfico de barras permite uma comparação visual imediata das métricas de Precisão e *Recall* ao longo das cinco iterações do processo de validação cruzada.

Esta representação visual evidencia que os resultados do *Web Service* implementado são consistentes com os do *framework Surprise* original, validando a corretude da implementação proposta.