Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Computação
Coordenação de Pós-Graduação em Ciência da Computação

Tese de Doutorado

# Explorando Características Sociais e de Colaboração na Recomendação de Projetos no GitHub

Thaciana Guimarães de Oliveira Cerqueira

Campina Grande, Paraíba, Brazil
22/04/2020

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

# Explorando Características Sociais e de Colaboração na Recomendação de Projetos no GitHub

Thaciana Guimarães de Oliveira Cerqueira

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Dr. Franklin de Souza Ramalho
Dr. Leandro Balby Marinho
(Orientadores)

Campina Grande, Paraíba, Brasil

C416e Cerqueira, Thaciana Guimarães de Oliveira.
Explorando características sociais e de colaboração na recomendação de projetos no GitHub / Thaciana Guimarães de Oliveira Cerqueira. – Campina Grande, 2020.
124 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.
"Orientação: Prof. Dr. Franklin de Souza Ramalho, Prof. Dr. Leandro Balby Marinho".
Referências.

1. Engenharia de Software. 2. Mineração de Dados. 3. Sistemas de Recomendação. 4. Aprendizagem de Máquina. 5. Plataforma GitHub. I. Ramalho, Franklin de Souza. II. Marinho, Leandro Balby. III. Título.

CDU 004.41(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

# EXPLORANDO CARACTERÍSTICAS SOCIAIS E DE COLABORAÇÃO NA RECOMENDAÇÃO DE PROJETOS NO GITHUB

**THACIANA GUIMARÃES DE OLIVEIRA CERQUEIRA**

**TESE APROVADA EM 22/04/2020**

**FRANKLIN DE SOUZA RAMALHO, Dr., UFCG**
**Orientador(a)**

**LEANDRO BALBY MARINHO, Dr., UFCG**
**Orientador(a)**

**CLÁUDIO ELÍZIO CALAZANS CAMPELO, PhD., UFCG**
**Examinador(a)**

**JOÃO ARTHUR BRUNET MONTEIRO, Dr., UFCG**
**Examinador(a)**

**LUCIANO DE ANDRADE BARBOSA, Dr., UFPE**
**Examinador(a)**

**FERNANDO JOSÉ CASTOR DE LIMA FILHO, Dr., UFPE**
**Examinador(a)**

**CAMPINA GRANDE - PB**

*Ao meu amado marido, Sérgio Cerqueira e minhas queridas filhas, Ingrid e Letícia Cerqueira.*

# Agradecimentos

Antes de tudo, agradeço a Deus por sempre está comigo, enchendo-me da sua graça, saúde, inteligência e inspiração para que fosse possível a realização desta pesquisa.

Agradeço ao meu marido Sérgio Cerqueira, que sempre me encorajou, dedicando muito amor, paciência e apoio para que eu pudesse realizar este trabalho. E, para minhas filhas amadas, Ingrid e Letícia, obrigada por todo o amor, incentivo e compreensão por tantos momentos de ausência de sua mãe tão ocupada nestes últimos anos.

Obrigada painho e mainha, Sebastião e Maria de Lourdes, por toda educação e constante motivação, que me permitiram crescer e realizar os meus sonhos. E, aos meus irmãos Neriglissor, Themístocles e Clístenes, e minha irmã Thessália, por todo o amor, parceria e por torcerem tanto pelo meu sucesso.

Sou muito grata aos meus orientadores Franklin Ramalho e Leandro Balby pela confiança, paciência e condução dedicada para este trabalho. Aprendi muito durante a orientação dos senhores, que levo não apenas para a minha tese, mas para toda a minha vida.

Sou grata aos professores João Arthur, Luciano Barbosa, Nazareno Andrade, Flavio Figueiredo, Cláudio Campelo e Fernando Castor pelos generosos comentários apresentados na versão preliminar deste documento. Tais comentários foram de grande importância na busca de uma unidade, coesão e rigor tanto na condução da pesquisa como na elaboração desta versão final.

Ao longo desta pesquisa, recebi sugestões de diversos colegas dos Laboratórios de Práticas de Software e Mineração de Dados, dentre os quais sou especialmente grata a Saulo Toledo, Mirna Maia, Alysson Milanez, Indy Paula, Matheus Oliveira, Melquisedec Andrade, Caio Santos, Ítalo Oliveira, Ricardo Oliveira, Allan Sales e demais colegas da sala 105.

Agradecimento especial para o Rafael Santos pelas muitas horas de discussão, sugestões e contribuições para este trabalho. Como também, para minha amada amiga Lilian Diniz por todo apoio, carinho e amizade para toda a vida.

Finalmente, por toda minha família e amigos, que me incentivaram, oraram e me apoiaram nesta jornada. Em especial, para minhas amigas Paula O'Neil e Valéria Santos, pelos conselhos e amizade.

# Resumo

A plataforma GitHub é o maior site de hospedagem de código-fonte, contendo um grande número usuários e repositórios de software. Devido a sua natureza social e colaborativa, os usuários são incentivados à contribuição nos repositórios uns dos outros. Neste contexto, é crucial compreender os fatores que representam os interesses dos usuários na plataforma, para que seja possível conceber serviços eficazes para ajudá-los na descoberta por repositórios relevantes. Sistemas de recomendação (SRs) têm sido cada vez mais populares para auxiliar os usuários na recuperação de informações e satisfação com os serviços online. SRs de repositórios na plataforma GitHub podem ser projetados para personalizar a experiência dos usuários e ajudá-los a descobrir projetos relevantes em um amplo e dinâmico espaço de busca. Entretanto, este é um cenário desafiador para recomendações. Desde a última década, métodos e técnicas de aprendizado de máquina têm sido explorados em pesquisas sobre SRs. Em geral, utilizam-se técnicas de classificação como Filtragem Colaborativa - FC e Filtragem Baseada em Conteúdo - FBC. A principal contribuição desta pesquisa é explorar o arcabouço de características presentes na plataforma GitHub e avançar neste conhecimento para recomendar projetos de interesse do usuário. Ela consiste na análise das diversas características de usuários e projetos extraídas dos dados e metadados disponíveis pela plataforma. Adicionalmente, propomos métricas geradas a partir dessas características para conhecermos das atividades dos projetos em termos de atratividade, visibilidade, medidas estatísticas e engajamento. Todas essas informações foram explicitamente modeladas através das técnicas FBC e FC (Item-KNN, Random Forest - RF, XGBoost e Máquina de Fatoração - MF), e aplicamos a recomendação dos repositórios para contribuição pelo usuário. Uma avaliação offline foi realizada considerando fatores que afetam o aprendizado da classificação, no qual sugerimos dois conjuntos de dados: amostragem balanceada e amostragem negativa, tendo um alcance de cerca de 500 mil projetos. Os experimentos implementados e executados nesta pesquisa mostram uma maior efetividade para as técnicas XGBoost e RF, em relação às técnicas Item-KNN, FBC e MF na recomendação de projetos nos aspectos de qualidade, como precisão e cobertura de projetos. Entre os resultados obtidos, XGBoost apresenta uma precisão 48% maior que Item-KNN na recomendação de projetos de interesse

do usuário. Entretanto, a precisão das recomendações de projetos não alcançaram 1% das $top-10$ recomendações. Por fim, propomos uma nova abordagem envolvendo o aprendizado da classificação, com o objetivo de aprimorarmos as recomendações por parte das técnicas aplicadas, onde obtivemos uma precisão com cerca de 40% dos projetos no top-10 da lista de recomendação. Concluímos que um melhor entendimento sobre a plataforma GitHub, no que diz respeito às características dos usuários e seus respectivos projetos, nos permitiram caracterizar os interesses dos usuários e implementar estratégias de recomendação de repositórios que correspondam às preferências dos usuários. Os resultados foram encorajadores, o que implica que as características propostas associadas às técnicas de aprendizagem de máquina representam as preferências do usuário. Esperamos que nossas descobertas pavimentem o caminho para a criação de novos sistemas de recomendação eficientes e algoritmos de busca para ajudar os usuários do GitHub a encontrar projetos de seu interesse.

# Abstract

The GitHub platform is the largest source code hosting site, containing a large number of users and software repositories. Due to their social and collaborative nature, users are encouraged to contribute to each other's repositories. In this context, it is crucial to understand the factors that represent the interests of users on the platform, so that it is possible to design effective services to assist them in discovering relevant repositories. Recommendation systems (SRs) have been increasingly popular to assist users in retrieving information and satisfaction with online services. SRs from repositories on the GitHub platform can be designed to personalize the user experience and help them discover relevant projects in a wide and dynamic search space. However, this is a challenging scenario for recommendations. Since the last decade, machine learning methods and techniques have been explored in research on SRs. In general, classification techniques such as Collaborative Filtering - FC and Content Based Filtering - FBC are used. The main contribution of this research is to explore the framework of features present on the GitHub platform and advance this knowledge to recommend projects of interest to the user. It consists of analyzing the various characteristics of users and projects extracted from the data and metadata available through the platform. Additionally, we propose metrics generated from these characteristics to learn about the activities of the projects in terms of attractiveness, visibility, statistics and engagement. All this information was explicitly modeled using the FBC and FC techniques (Item-KNN, Random Forest - RF, XGBoost and Factorization Machine - MF), and we applied the recommendation of the repositories for contribution by the user. An offline evaluation was carried out considering factors that affect the learning of the classification, in which we suggest two sets of data: balanced sampling and negative sampling, having a reach of about 500 thousand projects. The experiments implemented and performed in this research show greater effectiveness for the XGBoost and RF techniques, in relation to the Item-KNN, FBC and MF techniques in recommending projects in terms of quality, such as precision and project coverage. Among the results obtained, XGBoost presents an accuracy 48 % greater than Item-KNN in recommending projects of interest to the user. However, the

accuracy of the project recommendations did not reach 1 % of the $top - 10$ recommendations. Finally, we propose a new approach involving the learning of classification, with the objective of improving the recommendations by the applied techniques, where we obtained an accuracy with about 40 % of the projects in the top-10 of the recommendation list. We concluded that a better understanding of the GitHub platform, with regard to the characteristics of users and their respective projects, allowed us to characterize the interests of users and implement strategies for recommending repositories that correspond to users' preferences. The results were encouraging, which implies that the proposed characteristics represent the user's preferences. We hope that our findings will pave the way for the creation of new efficient recommendation systems and search algorithms to help GitHub users find projects that interest them.

# Conteúdo


**1 Introdução** **1**

1.1 Motivação . . . . . 2

1.2 Definição do Problema . . . . . 7

1.3 Escopo da Pesquisa . . . . . 9

1.4 Objetivos e Questões de Pesquisa . . . . . 10

1.5 Estrutura do Documento . . . . . 11

**2 Fundamentação Teórica** **13**

2.1 Sistemas de Recomendações . . . . . 13

2.2 Métodos de Classificação . . . . . 16

2.2.1 Filtragem Baseada em Conteúdo . . . . . 16

2.2.2 Filtragem Colaborativa (FC) . . . . . 18

2.2.3 Filtragem Híbrida (FH) . . . . . 25

2.3 Redes Sociais . . . . . 26

2.4 GitHub . . . . . 28

2.5 Considerações Finais . . . . . 31

**3 Trabalhos Relacionados** **33**

3.1 Ecossistema GitHub . . . . . 33

3.2 Aplicações de Recomendação no GitHub . . . . . 39

3.3 Considerações Finais . . . . . 43

**4 Exploração de Dados do GitHub** **45**

4.1 Coleta dos Dados . . . . . 45

4.2 Análise dos Dados . . . . . 46

4.3 Considerações Finais . . . . . 54

**5 Metodologia Aplicada** **55**

5.1 Visão Geral . . . . . 56

5.2 Pré-Processamento de Dados . . . . . 58

5.2.1 Amostragem . . . . . 59

5.3 Seleção das Características . . . . . 62

5.4 Atributos Derivados e Métricas . . . . . 66

5.5 Métodos de Classificação . . . . . 69

5.5.1 Filtragem Baseada em Conteúdo . . . . . 69

5.5.2 FC Baseada em Vizinhança - Item-KNN . . . . . 72

5.5.3 Árvore de Decisão . . . . . 73

5.5.4 Máquina de Fatoração . . . . . 76

5.6 Métricas . . . . . 77

5.7 Considerações Finais . . . . . 79

**6 Análise e Resultados** **80**

6.1 Validação da Qualidade da Recomendação . . . . . 80

6.2 Efeito das Características no Aprendizado dos Modelos . . . . . 86

6.3 Nova Proposta de Aprendizado Para Classificação . . . . . 96

6.4 Ameaças à Validade . . . . . 101

6.5 Considerações Finais . . . . . 102

**7 Conclusões** **104**

7.1 Contribuições . . . . . 106

7.2 Limitações e Trabalhos Futuros . . . . . 107

**A Comparação de *F-Measure*** **123**

# Lista de Figuras


1.1 Exemplo do ambiente GitHub - Repositórios Descobertos . . . . . 5

2.1 Ilustração dos elementos envolvidos em um SR . . . . . 14
2.2 Taxonomia dos Métodos de Sistemas de Recomendação . . . . . 15
2.3 Fatoração matricial para um sistema de recomendação. Usuários e itens são caracterizados por seus respectivos vetores fatoriais k-dimensionais. . . . . 21
2.4 Vetor de entrada para um sistema de recomendação. . . . . 22
2.5 Exemplo de uma rede social de amizade . . . . . 26
2.6 Exemplo do ambiente GitHub - Perfil . . . . . 29
2.7 Simplificação do Modelo Baseado em *Pull* . . . . . 30

3.1 Principais Contribuições no GitHub (imagem apresentada na pesquisa de Geiger [39]) . . . . . 34
3.2 Dificuldades Identificadas no Processo *Open Source* (imagem apresentada na pesquisa de Geiger [39]) . . . . . 35
3.3 Usuários estão muito interessados em realizar contribuições futuras (lado esquerdo). Entretanto, não estão muito propensos a realizar tais contribuições (lado direito). (imagem apresentada na pesquisa de Geiger [39]) . . . . . 37
3.4 Usuários utilizam como principal meio de comunicação os fóruns, tanto para pedir ajuda (lado esquerdo) como para ajudar (lado direito). (imagem apresentada na pesquisa de Geiger [39]) . . . . . 37
3.5 A maioria dos contribuidores não tinham um relacionamento prévio com os usuários os quais estavam ajudando (lado esquerdo) ou sendo ajudados (lado direito) (imagem apresentada na pesquisa de Geiger [39]) . . . . . 38
3.6 Quantidade de *Pull-Requests* por Ano . . . . . 39

4.1 Linguagens de Programação Mais Utilizadas . . . . . 49
4.2 Linguagens de Programação com mais *commits* . . . . . 50
4.3 *Commits* Diários dos Projetos . . . . . 50
4.4 Crescimento da quantidade de *pull-requests* por ano . . . . . 53
4.5 Comparação dos repositórios base e repositórios *fork* em relação à quantidade de *commits* e *pull-requests* . . . . . 54

5.1 Visão Geral . . . . . 57
5.2 Estrutura de um projeto na linha do tempo. . . . . 67
5.3 Exemplo de relacionamentos, por exemplo, entre usuários e projetos . . . . 73
5.4 Modelo de conjunto de árvores. A previsão final para um determinado exemplo é a soma das previsões de cada árvore. . . . . 74
5.5 A estrutura exibe a soma do gradiente estatístico para cada nó folha e define o *score*. . . . . 76

6.1 Resultados de *Precision* para Todos os Modelos . . . . . 83
6.2 Resultados de *Recall* para Todos os Modelos . . . . . 83
6.3 Comparação entre Modelos usando F-Measure . . . . . 84
6.4 Linguagens de Programação dos Projetos Mais Recomendados . . . . . 87
6.5 Resultado da execução das características mais relevantes para o modelo XGBoost (*weight*) . . . . . 89
6.6 Resultado da execução das características mais relevantes para o modelo XGBoost (*cover*) . . . . . 90
6.7 Resultado da execução das características mais relevantes para o modelo treinado XGBoost (*gain*) . . . . . 90
6.8 Alteração de magnitude média Shap . . . . . 91
6.9 Alteração de Magnitude Média (*Tree SHAP*) . . . . . 92
6.10 Relação de proximidade entre as características . . . . . 93
6.11 Comparação entre Modelos usando F-Measure para 10 (dez) Características 95
6.12 Predições para o Modelos . . . . . 96
6.13 Resultados da Métricas para Aprendizado da Classificação . . . . . 100

6.14 Posições das Recomendações para o Modelos na Nova Proposta para o Aprendizado da Classificação . . . . . 101

# Lista de Tabelas


2.1 Matriz Esparsa representando Feedback Explícito . . . . . 23
2.2 Matriz Esparsa representando Feedback Explícito . . . . . 24
2.3 Conceitos do GitHub . . . . . 32

3.1 Estudos sobre Sistemas de Recomendação no GitHub . . . . . 40

4.1 Organizações (O) e Repositórios (P) com seus Eventos por Contribuidores . 47
4.2 Repositórios com maior número de *forks* até abril de 2018 . . . . . 51

5.1 Características utilizadas pelos métodos de recomendação. Característica (CRT) Explícita (EX) ou Implícita (IM) . . . . . 70
5.2 Modelo de Máquina de Fatoração . . . . . 77

6.1 Amostra das Características Semelhantes entre Usuário e Projeto . . . . . 81
6.2 Amostra das Características no Conjunto de Treino . . . . . 82
6.3 *Precision*(P), *Recall*(R), *F-Measure*(F-M) para Amostragem Balanceada e Negativa . . . . . 85
6.4 Top-10 projetos recomendados utilizando XGBoost . . . . . 88
6.5 *Precision*(P), *Recall*(R), *F-Measure*(F-M) para Amostragem Negativa . . . 94
6.6 *Precision*(P), *Recall*(R), *F-Measure*(F-M) . . . . . 99

A.1 Melhores Resultados dos Modelos para F-Measure para QP1 (FM-1), QP2 (FM-2) e QP3 (FM-3) . . . . . 124

# Capítulo 1

# Introdução

Desde meados de 2000, os Sistemas de Controle de Versão Distribuído (SCVDs) têm sido amplamente utilizados, aumentando sua popularidade em relação aos tradicionais Sistemas de Controle de Versão Centralizado (SCVCs). Os SCVCs disponibilizam uma série de recursos que são muito utilizados pelos desenvolvedores de sofware, entretanto, a nova metodologia dos SCVDs atraiu atenção e adoção generalizadas [81]. Os SCVDs possibilitam que os desenvolvedores (i) trabalhem isoladamente e de forma *offline* com cópia local dos repositórios, mantendo o histórico completo de alteração; (ii) criem e mesclem ramificações durante o desenvolvimento; e (iii) confirmem mudanças para qualquer tipo de arquivo, sejam apenas poucas linhas ou arquivos inteiros, dando habilidade total de acompanhamento das revisões para os desenvolvedores de sofware [18], além de integrar vários recursos de codificação social.

Nos últimos anos, a codificação social tornou-se uma importante abordagem colaborativa para o desenvolvimento de software de código aberto. A codificação social é uma abordagem de desenvolvimento de software que enfatiza a colaboração formal e informal, permitindo uma experiência diferente de desenvolvimento de software, onde as atividades e os interesses de um desenvolvedor são facilmente visualizadas por outros desenvolvedores [105]. Vários sistemas maduros como Bazaar[1], Mercurial[2] e Git[3] apareceram combinados com as redes

[1] http://bazaar.canonical.com

[2] https://www.mercurial-scm.org/

[3] https://git-scm.com/

sociais Launchpad[4], Bitbucket[5] e GitHub[6] [79]. O GitHub é uma das plataformas mais populares de controle de versão colaborativa para desenvolvedores e, sem dúvida, o maior site de hospedagem de código do mundo [73].

A rede social GitHub contém cerca de 24 milhões de usuários cadastrados e hospeda mais de 83 milhões de repositórios (GitHubArchive[7]: Ano 2019). Para a codificação social, projetos de software também são chamados de repositórios ou, simplemente, projetos. O repositório pode conter um conjunto de artefatos principais (por exemplo, documentos de texto, código-fonte, imagens), além de outras fontes de dados associadas a eles, como as características sobre o seu usuário proprietário. Então, a variedade de repositórios na plataforma vai desde o kernel do Linux e famosos editores de código-fonte como Visual Studio Code (VS Code) e Vim, até projetos, por exemplo, não relacionados a software como os modelos de trabalhos científicos da Universidade de Brasília - UNB[8]. Portanto, o GitHub tornou-se uma fonte rica de conteúdo, onde existem dois tipos de entidades principais na plataforma: usuário e repositório.

## 1.1 Motivação

Nas últimas décadas, muitos estudos [4] [55] [48] [30] dedicaram-se ao incentivo à codificação social para obter maior desempenho e contribuição entre os membros dos projetos. Seja por meio das organizações de trabalho ou por ferramentas sociais de colaboração, a proveitosa contribuição entre membros de uma equipe é a base para impulsionar a inovação e o sucesso de um projeto. Como resultado, muito se avançou a respeito da criação de plataformas de hospedagem de códigos como Sourceforge[9], CodePlex[10], Google Code[11], IBM Bluemix [12] e GitHub [5].

No GitHub, a codificação social é estimulada entre os usuários, dentro e entre os pro-

[4]https://launchpad.net/bzr
[5]https://bitbucket.org/
[6]https://github.com
[7]https://www.gharchive.org/
[8]https://github.com/UnB-CIC
[9]https://sourceforge.net/
[10]https://www.codeplex.com
[11]https://code.google.com
[12]https://developer.ibm.com/bluemix/

jetos, que podem ser criados publicamente e facilmente contribuídos por colaboradores via interface transparente e independente de localização geográfica. A plataforma disponibiliza informações dos usuários que vão desde as organizações que participam, localização geográfica, quantidade de *followers* (seguidores), até informações sobre os projetos (descrição textual, data de criação, quantidade de *watchers* (observadores), linguagem de programação utilizada, histórico de *commits*, *pull-requests*, *issues* e outros) - os termos aqui apresentados encontram-se descritos no Capítulo 2. Este ambiente transparente é um aspecto valioso da comunidade GitHub [46].

Dabbish et al. [32] cita que a visibilidade de algumas ações do GitHub, como seguir um desenvolvedor ou observar um projeto, leva ao aumento da qualidade das atualizações dos projetos por parte dos desenvolvedores (por exemplo, o desenvolvedor mantém-se motivado pelo simples fato de saber que alguém se interessou pelo seu trabalho). Além disso, essas ações levam ao aumento das extensões desses projetos (*fork*) por outros desenvolvedores, à exposição de outros projetos do portfólio do usuário, à geração de novas ideias, debates e contribuições de projetos entre os usuários.

Entretanto, a grande quantidade de conteúdo gerado na plataforma representa um desafio em termos de sobrecarga de informações. Normalmente, motores de busca são disponibilizados para facilitar a descoberta de repositórios, como também são utilizados Sistemas de Recomendação (SRs). Os motores de busca tiveram um impacto significativo na última década e são eficazes na filtragem de páginas na web que correspondem a consultas explícitas pelo usuário [93]. Tornou-se, portanto, uma atividade dominante na web, mas, ainda assim, é difícil julgar a relevância e classificar os resultados dessas consultas apenas com os dados fornecidos pelo usuário.

Os SRs ganharam uma atenção significativa a partir de 2006, quando a empresa NetFlix[13] lançou o *Prêmio NetFlix* para melhorar as recomendações de filmes [10]. Os SRs são ferramentas e técnicas de software que fornecem sugestões de itens baseadas no perfil do usuário alvo (usuário ao qual destina-se a recomendação) [92]. As sugestões podem ser, por exemplo, quais itens comprar, que música ouvir ou em qual projeto de software contribuir. Os casos de sucesso no uso de soluções baseadas em SRs, como, por exemplo, Netflix, Fa-

[13] `http://www.netflix.com`

cebook[14], Amazon[15] e Youtube[16], têm motivado novas iniciativas no emprego de sistemas de recomendação nos mais diversos contextos e provaram ser um meio valioso para lidar com o problema de sobrecarga de informações, quando os usuários são submetidos em excesso à informação. [93].

A plataforma GitHub oferece tanto o serviço de motor de busca, como também um sistema de recomendação para descoberta de repositórios. Mesmo com a disponibilidade de um motor de busca na plataforma, os usuários esbarram na dificuldade de encontrar repositórios de seu interesse para contribuição [12]. Uma das formas encontradas pelos usuários para contornar esta dificuldade, foi a criação de repositórios que informam listas de projetos separados por linguagem de programação, inspirados no *First Time Only*[17], utilizado, principalmente, por iniciantes que desejam conhecer sobre os projetos e realizar contribuições. A Figura 5.1[18] exibe a descoberta de repositórios disponibilizado pelo GitHub, que baseia-se no comportamento do usuário em relação aos projetos que este considera como relevantes para observar, suas relações com outros usuários para identificar as preferências deste usuário [41] e na classificação das estrelas dos repositórios baseados nos interesses dos usuários.

O sistema de recomendação provido pelo GitHub tem motivado novas iniciativas de recomendação de repositórios utilizando outras características disponibilizadas pela plataforma. Encontramos alguns estudos que assumiram a tarefa de recomendar repositórios para os usuários ou revisões de código-fonte de projetos do GitHub. Alguns deles, utilizaram a linguagem de programação como base para as recomendações de repositórios [58] [65], outros combinaram as informações dos usuários com as informações de atualização dos projetos para recomendação de revisores de código para os repositórios [68] [115] [103] [113] [118]. Esses estudos propõem mitigar os esforços dos desenvolvedores de software em encontrar repositórios para contribuição.

O desafio de criar experiências personalizadas para os usuários por meio das técnicas de sistemas de recomendação é grande. Historicamente, as pesquisas focadas em SRs concentram-se, principalmente, em cenários em que as escolhas anteriores dos usuários es-

[14] `https://www.facebook.com`
[15] `http://www.amazon.com`
[16] `http://www.youtube.com`
[17] https://blog.kentcdodds.com/first-timers-only-78281ea47455
[18] `http://www.github.com`

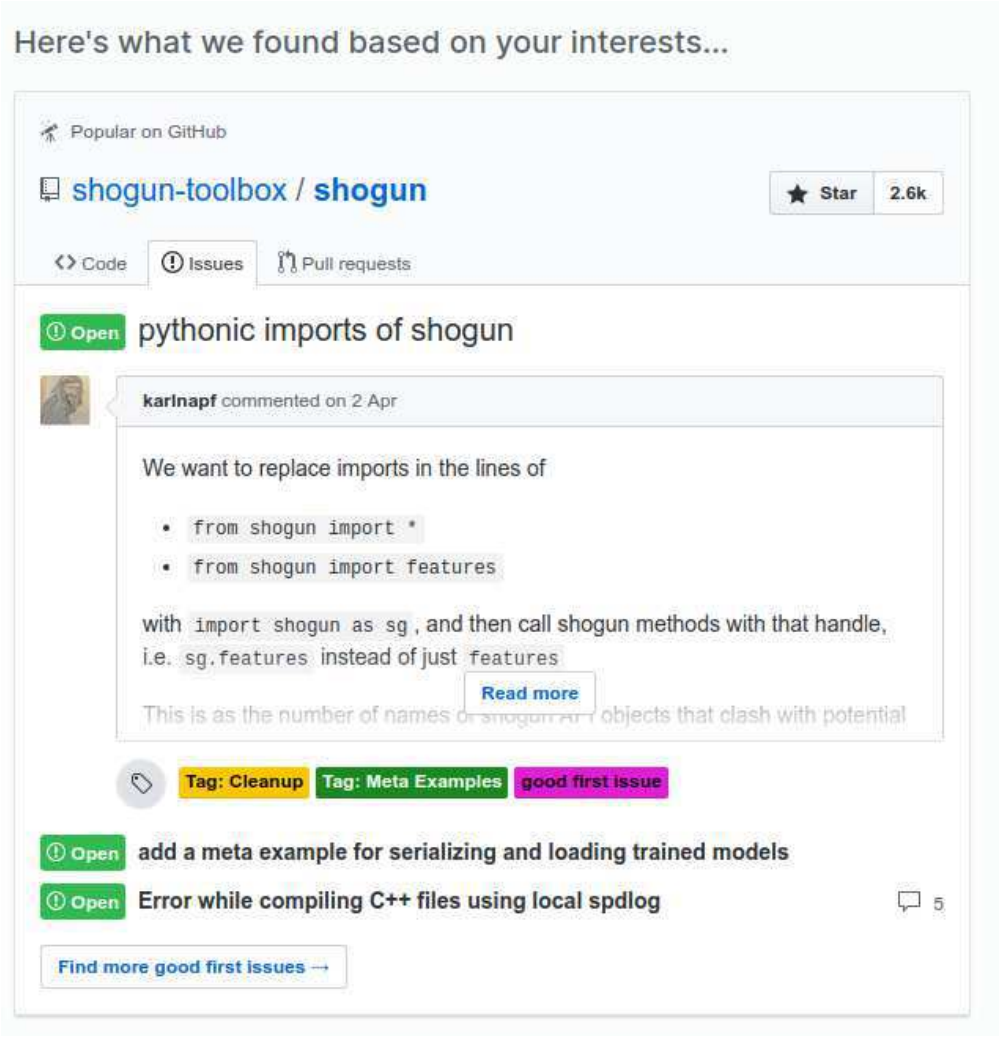


Figura 1.1: Exemplo do ambiente GitHub - Repositórios Descobertos

tão disponíveis, possibilitando modelar perfis de usuários e, neste contexto, também perfis de projetos, de forma precisa e com o intuito de realizar recomendações significativas. Analisando os estudos sobre recomendações de projetos na plataforma GitHub, identificamos uma oportunidade de pesquisa na investigação de modelos de recomendação no domínio de repositórios do GitHub utilizando uma diversidade de características disponibilizadas pela plataforma e a criação de novas características derivadas a partir das existentes. Escolhemos a plataforma GitHub devido a sua grande utilização em todo o mundo, mas o trabalho realizado nesta pesquisa pode ser aplicado às demais plataformas de colaboração que se utilizam de recursos de redes sociais e que contenham as características semelhantes aplicadas neste estudo.

Uma característica de grande relevância para o nosso trabalho de recomendação de repo-

sitórios foi o uso do recurso conhecido por *fork* (cópia de um repositório para contribuições), que propomos como classe binária para aplicação aos modelos de classificação. A informação de um projeto ser *fork* ou não de um usuário, nos permite conhecer sobre o interesse de contribuição deste usuário na plataforma. Identificamos que, dentre os 83 milhões de repositórios, apenas 19,7% dos repositórios são *fork* de outros projetos. Ou seja, a contribuição social tem muito a crescer ainda no GitHub e o uso dos SRs são de grande relevância neste ambiente. No estudo de Gousios et al. [43], cita que o *fork* é uma das principais características da plataforma GitHub que demonstra que um projeto receberá uma contribuição, além do uso de *pull-request*. É condição, dentro da plataforma, que para se aplicar um *pull-request* é preciso realizar um *fork*. Dentro deste contexto, decidimos utilizar o *fork* como a principal informação em relação a um repositório receber uma contribuição ou não por um usuário.

Assim, encontrar os repositórios certos para o usuário participar [19] é crucial no estímulo à codificação social. Como dito anteriormente, esse ambiente transparente é um aspecto valioso da comunidade GitHub, não apenas para manter os contribuidores estimulados à codificação social e atualizados sobre o status dos projetos [44], como também, por termos acesso aos dados disponibilizados pela plataforma para gerarmos SRs precisos e adequados ao ambiente GitHub.

Desta forma, visualizamos como uma oportunidade de pesquisa a exploração das características sociais e de colaboração na recomendação de projetos no GitHub. Nesse cenário, propomos uma abordagem baseada em aprendizado de máquina para tratar a dificuldade dos usuários de encontrarem repositórios para contribuição na plataforma GitHub devido (i) à grande quantidade de conteúdo produzido e que representa um desafio em termos de sobrecarga de informações para o usuário. Como também, (ii) ao pouco conhecimento sobre as características dos usuários e projetos, suas derivações e seu comportamento em um ambiente de recomendação e de como esse conhecimento pode ser útil na aplicação de recomendadores.

[19] *Participar* seria observar, realizar *commits*, *pull-requests* e fomentar ideias ou esclarecer dúvidas através de *issues*.

## 1.2 Definição do Problema

Neste trabalho, propomos um recomendador multi-contextual onde o problema que buscamos solucionar pode ser especificado como segue: dado um usuário alvo do GitHub e um conjunto de dados contextuais, quais repositórios tem maior probabilidade do usuário realizar um *fork*?

O processo de recomendação geralmente começa com a especificação de um conjunto inicial de classificação que é explicitamente fornecido pelos usuários ou é implicitamente inferido pelo sistema. Uma vez especificadas essas classificações iniciais, um sistema de recomendação tenta estimar a função de classificação $R$ para os pares (usuário, item) que ainda não foram classificados pelos usuários.

$$R : Usuario \times Item \rightarrow Classe$$

A classe é um conjunto ordenado (por exemplo, número binário ou números reais dentro de um determinado intervalo), e $Usuario$ e $Item$ são os domínios de usuários e itens, respectivamente.

No nosso estudo, a função de classificação é representada pelos domínios de usuário (que são representados pelos usuários do GitHub) e itens (que são representados pelos projetos criados na plataforma GitHub), e a classe binária é representada pela característica *fork* (0 para o projeto que não é *fork* e 1 para o projeto que é *fork*). Portanto, a função de classificação $R$ determina para os pares (usuário, projeto), que ainda não foram classificados pelos usuários, se o par seria um *fork* ou não.

$$R : Usuario \times Projeto \rightarrow Fork$$

Ou seja, neste caso, o usuário não classifica se um projeto é *fork* ou não, a plataforma GitHub informa se o projeto recebeu o evento *fork* ou não. Nós capturamos essa informação para o aprendizado do recomendador e a função classificação estima se os projetos candidatos poderão ser um *fork* ou não a partir do contexto dos usuários e projetos.

Muitas das pesquisas realizadas na área de sistemas de recomendação, concentram-se em recomendar os itens mais relevantes aos usuários sem levar em consideração informações contextuais adicionais, como, por exemplo, tempo, localização e ambiente social [3]. Esses

SRs tradicionais modelam interações apenas com os dois tipos de entidades, usuários e itens, não os colocando em um contexto ao fornecer as recomendações.

O conceito de contexto foi estudado por várias disciplinas, onde cada uma delas tende a ter sua própria definição e não apenas utilizar a definição genérica padrão de contexto que encontramos no dicionário: "condições ou circunstâncias que afetam alguma coisa" [110]. A inclusão das informações contextuais no processo de recomendação apresenta oportunidades para interações mais ricas entre usuários e itens. No nosso estudo, o contexto é focado nas circunstâncias que se caracterizam pelas informações de entrada do sistema de recomendação, onde encontramos os diferentes status, preferências e contribuições sobre os usuários e projetos. Como exemplo de contexto em relação ao status, preferências e contribuições de usuários e projetos na plataforma GitHub pode ser: adicionar um *issue* em um projeto, seguir um novo usuário ou realizar um *commit* em um projeto. Deste modo, uma das principais tarefas no processo de recomendação contextual em um ambiente virtual de colaboração é a etapa de modelagem dos perfis dos usuários e projetos.

Formalizando, além do conjunto de usuários $U$ e do conjunto de projetos $P$, consideramos os dados contextuais do conjunto de atratividade $A$, que são os fatores relacionados à popularidade dos usuários e projetos; do conjunto de visibilidade $V$, disponibilizado pela plataforma; do conjunto de engajamento $E$, que representa a atividade do projeto; e, do conjunto de distribuição dos dados $D$, que são medidas estatísticas aplicados às características. Consideramos dados explícitos e implícitos, onde temos o conjunto $S \subseteq U \times P \times A \times V \times E \times D$ das relações entre *usuários*, *projetos*, *atratividade*, *visibilidade*, *engajamento* e *medidas estatísticas*. A tarefa do recomendador de projetos é então encontrar uma função que atribua um valor de preferência para os projetos candidatos.

$$\hat{s} \subseteq U \times P \times A \times V \times E \times D \rightarrow \mathbb{R}$$

Assim, dado um usuário alvo $u \in U$ e os sinais contextuais $a \in A$, $v \in V$, $e \in E$ e $d \in D$, as top-n recomendações podem ser computadas por

$$\text{top}N(u, a, v, e, d) := \underset{p \in |P_c| \setminus \{|P_u|\}}{\overset{N}{\text{argmax}}} \hat{s}(u, p, a, v, e, d),$$

onde $N$ denota o número de projetos a serem recomendados, $P_c$ os projetos candidatos e $P_u$ os projetos que o usuário $u$ realizou *forks* no passado.

Em cenários de recomendação $top - N$, o objetivo é minimizar a função de custo $\ell$ que mede a diferença entre a lista de recomendação (gerada por $\hat{s} \in \mathbb{R}^P$) em relação à lista real de projetos (onde $\mathcal{P}(P)$ é um sub-conjunto de $P$, ou seja, são os projetos do usuário dentro do conjunto $P$) dos usuários, que encontram-se no conjunto de teste e que são desconhecidos durante o treino.

$$\ell : \mathcal{P}(P) \times \mathbb{R}^P \to \mathbb{R}$$

Com isso, utilizamos $\hat{s}_W(u, p)$ para indicar a recomendação de projeto que calcula a pontuação de relevância do projeto $p \in P$ para o usuário $u \in U$ com base no contexto $W$.

## 1.3 Escopo da Pesquisa

Historicamente, as pessoas contam com recomendações e menções de seus pares ou com o aconselhamento de especialistas para apoiar decisões e descobrir novas informações sobre determinado assunto. Esses métodos de recomendar coisas novas são limitados. Por exemplo, pode haver um filme ou livro que uma pessoa goste, mas ninguém em seu círculo de amizade tenha ouviu falar dele. Os SRs utilizam-se do poder computacional e oferecem a oportunidade de expansão do conjunto de pessoas das quais os usuários podem obter recomendações. Eles também permitem a mineração do histórico das informações sobre usuários e itens, potencialmente fornecendo uma experiência de seleção mais precisa [36].

Esses SRs se beneficiam dos resultados obtidos em vários campos da ciência da computação, especialmente aprendizado de máquina e mineração de dados. Eles permitem que o computador aprenda a executar de maneira otimizada uma determinada tarefa utilizando-se de exemplos, dados ou experiências passadas [102]. Consequentemente, as recomendações podem ser construídas usando as informações fornecidas por essas associações. Muitos SRs estão centrados no uso de vários algoritmos de aprendizado de máquina e de mineração de dados para prever avaliações do usuário para itens ou para aprender a classificar corretamente os itens para um usuário [93].

Basicamente, o processo de mineração de dados apresenta 3 etapas: pré-processamento, análise de dados e avaliação dos resultados. O usuário é o ponto central do processo, iniciando a busca pela informação de acordo com o seu interesse. No nosso estudo, aplicamos

a recomendação de projeto offline, ou seja, passamos pelo pré-processamentos e análise dos dados, chegando a recomendação dos projetos, mas não aplicamos a avaliação da recomendação diretamente pelo usuário final, chamada de recomendação on-line. Nesta, é recebido o feeedback do usuário em relação as recomendações propostas pelo recomendador. Entretanto, apesar de não apresentarmos uma recomendação on-line, geramos a recomendação offline, utilizando dois grandes conjuntos de dados gerados a partir de dados temporais offline. Recuperamos dados de usuários e projetos no período de tempo de 05 (cinco) anos (cerca de 10 mil usuários e 500 mil projetos), pois seria inviável, por questões de processamento, analisar todos os projetos da plataforma desde sua criação (atualmente a plataforma encontra-se com cerca de 83 milhões de projetos).

Portanto, o escopo desta pesquisa é analisar e avaliar as diversas características de usuários e projetos extraídas dos dados disponíveis pela plataforma GitHub. Para melhor compreensão de cada tipo de característica e enquadramento das informações, separamos as mesmas em grupos que chamamos de *atratividade*, *visibilidade*, *engajamento* e *medidas estatísticas*. Os dois primeiros grupos contêm características capturadas diretamente da plataforma GitHub. Quanto ao terceiro e quarto grupos, as características foram derivadas a partir de eventos gerados por ações dos usuários no uso da plataforma. Essas características são parte da contribuição do nosso trabalho, onde não identificamos nenhum outro trabalho que tenha criado características semelhantes para serem utilizadas em um sistema de recomendação. Além disso, aplicamos algoritmos de aprendizagem de máquina para avaliarmos a qualidade das recomendações de projetos a partir das características selecionadas.

## 1.4 Objetivos e Questões de Pesquisa

O principal objetivo deste trabalho é avaliar a qualidade das recomendações de projetos geradas a partir das predições realizadas pelos modelos de classificação aplicados às diversas características de usuários e projetos extraídas dos dados disponibilizados pela plataforma GitHub.

Os objetivos específicos para investigação deste trabalho são apresentados a seguir:

- Analisar a plataforma GitHub e investigar as características possíveis de serem aplicadas em um sistema de recomendação;

- Propor novas características derivadas a partir das características selecionadas explícita ou implicitamente;
- Empregar e avaliar o desempenho de diferentes técnicas de aprendizado de máquina com o intuito de classificar os repositórios com base nos dados de usuários e projetos disponíveis na plataforma GitHub;
- Comparar as diferentes técnicas de aprendizado de máquina selecionadas aplicadas aos SRs;
- Avaliar as Questões de Pesquisa declaradas nos conjuntos de dados selecionados e analisar os fatores de qualidade das recomendações.

As Questões de Pesquisa (QP) abordadas por esta tese são apresentadas a seguir:

- QP1 - Como atuam os sistemas de recomendação no domínio de projetos do GitHub em termos de fatores de qualidade das recomendações?
- QP2 - Quais são as características, dentre os conjuntos de dados de atratividade, visibilidade, engajamento e medida das estatísticas, que se mostram mais relevantes em termos de fatores de qualidade das recomendações de projetos?
- QP3 - Qual o efeito nos fatores de qualidade das recomendações, quando aplicada uma nova metodologia de teste a partir da amostragem negativa?

## 1.5 Estrutura do Documento

Os conceitos tratados em Sistemas de Recomendação envolvem estudos em diversas áreas. O Capítulo 2 tem como objetivo apresentar a contextualização dos estudos sobre Sistemas de Recomendação e destacar os principais conceitos do GitHub e Redes Sociais, que são relevantes à pesquisa descrita neste documento. No Capítulo 3, apresentamos a análise dos trabalhos relacionados. Nessa análise, dá-se ênfase aos trabalhos que tratam sobre Sistemas de Recomendação para Redes Sociais e para o ambiente do GitHub, procurando abranger a maioria dos estudos sobre recomendação para diversas entidades do GitHub. Em relação à

parte descritiva dos dados e conceitos utilizados no processo de contribuição e colaboração na plataforma GitHub, detalhamos essas informações no Capítulo 4.

Após essa análise conceitual, no Capítulo 5, apresentamos a metodologia aplicada para recomendação de projetos, que apresenta o arcabouço teórico-conceitual utilizado, as abordagens propostas e os métodos de avaliação utilizados. No Capítulo 6, desenvolvemos a análise dos principais resultados experimentais e a discussão das Questões de Pesquisa. Finalmente, no Capítulo 7 destacamos as principais conclusões e contribuições da pesquisa descrita neste documento. Além disso, discutemos novas perspectivas de trabalhos futuros que se abrem a partir da pesquisa reportada neste documento.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo, introduzimos os sistemas de recomendação tradicionais (Seção 2.1), as técnicas de aprendizagem de máquina utilizadas para a classificação (Seção 2.2) , as redes sociais (Seção 2.3) e, por fim, discutimos sobre alguns estudos realizados com o intuito de entender o comportamento dos usuários na plataforma GitHub (Seção 2.4).

## 2.1 Sistemas de Recomendações

Em meados da década de 90, os sistemas de recomendação emergem como uma alternativa aos estudos clássicos em sistemas de informação, tendo como principal objetivo a diminuição do espaço de busca do usuário, auxiliando-o na tomada de decisão. Os SRs fundamentam-se na observação de que a maioria das pessoas confiam nas recomendações fornecidas por outros para tomar decisões cotidianas [92]. Assim, SRs são aplicações de software que visam dar suporte aos usuários para tomada de decisão fornecendo recomendações de itens baseados nas preferências destes usuários, enquanto interagem com uma grande quantidade de informação.

Em geral, os três tipos de elementos relacionados a uma proposta de recomendação são usuário, item e transação [93], como definidos abaixo e ilustrados pela Figura 2.1.

- **Usuário**: Os usuários de um sistema de recomendação podem ter objetivos e características muito diferentes. Para personalizar as recomendações durante a interação homem-máquina, os SRs exploram uma gama de informações e dados sobre os usuários. Essas informações podem ser estruturadas de várias formas e a seleção de quais

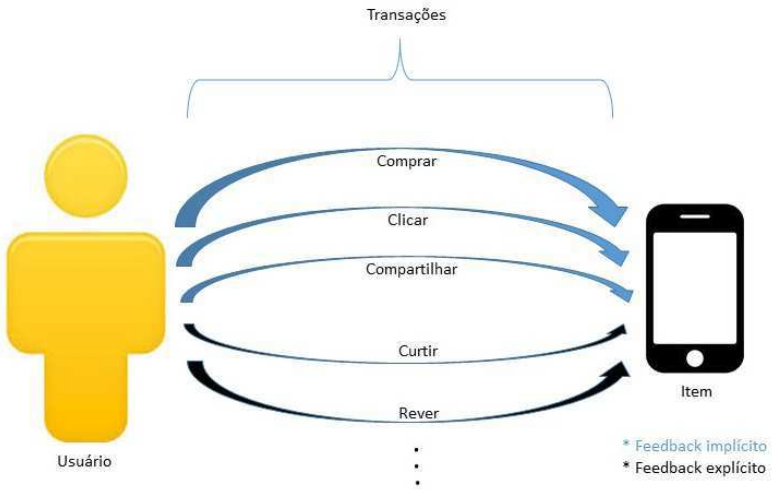


Figura 2.1: Ilustração dos elementos envolvidos em um SR

informações serão utilizadas depende da técnica de recomendação. Esse conjunto de preferências do usuário é chamado de **perfil de usuário**, podendo ser composto, por exemplo, com as informações de idade, sexo e profissão [93];

- **Item**: Termo genérico usado para identificar cada um dos elementos recomendados pelos SRs (ex: filmes, músicas, livros, restaurantes, etc.) e pode ser caracterizado pela sua complexidade, valor ou utilidade. O valor de um item pode ser positivo (comprou um filme, favoritou um música, gostou de uma publicação) ou negativo (removeu um link dos favoritos, desgostou de uma publicação). Dessa forma, os SRs podem usar uma variedade de propriedades e características dos itens, por exemplo, a descrição de um filme, onde temos: o gênero, o nome do diretor, os atores que participam do filme. Esse conjunto de informações são as características (*features*) do item, o qual chamamos de **perfil do item** [93];

- **Transação**: É uma interação registrada entre um usuário e o sistema de recomendação. Várias transações geram um conjunto de dados que contém informações relevantes e que são disponibilizadas durante a interação homem-máquina. Os SRs coletam as preferências do usuário por meio dessas transações, que podem ser expressadas explicitamente (o usuário quantifica a sua preferência pelo item) ou implicitamente (eventos que indicam, mas não quantificam a preferência do usuário sobre um determinado

item). As preferências do usuário são úteis para gerar as recomendações [93].

As principais famílias de métodos de recomendação são Filtragem Colaborativa (FC), Filtragem Baseada em Conteúdo (FBC) e Filtragem Híbrida (FH), conforme mostrado na Figura 2.2, que são brevemente descritos na próxima subseção.

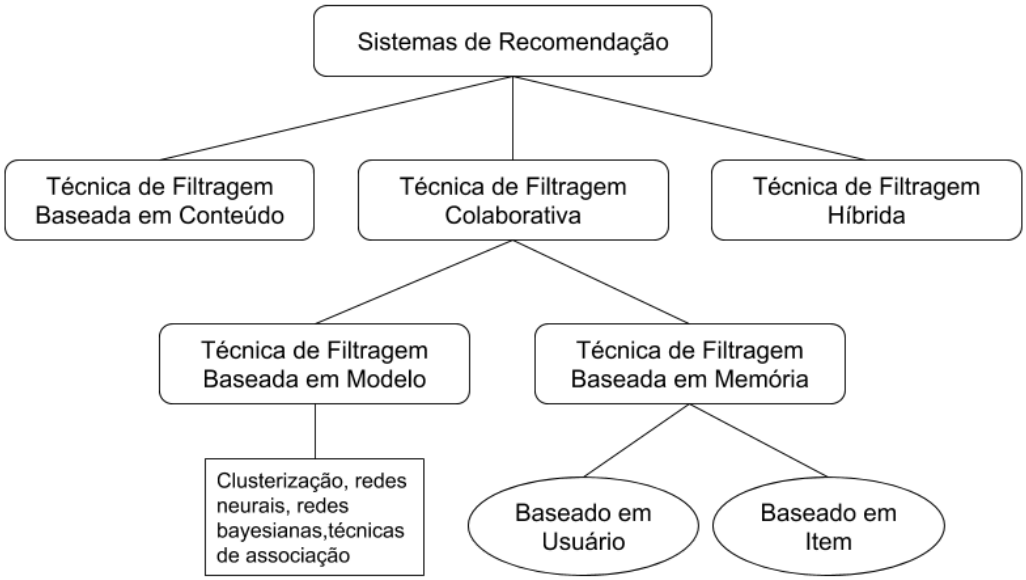


Figura 2.2: Taxonomia dos Métodos de Sistemas de Recomendação

Algumas técnicas de recomendação mais simples sugerem itens mais populares para um usuário de forma não personalizada. Por exemplo, o top-10 *best-seller* de filmes de terror. Esta recomendação é aplicável para os usuários, uma vez que, uma possível solução é recomendar a média geral das preferências dos usuários aplicados ao contexto. Neste caso, os itens mais populares são mais propensos de apreciação do que um item aleatório [2] [92].

Esta pesquisa é especialmente focada na filtragem colaborativa (que captura comportamentos anteriores do usuário e item) e filtragem baseada em conteúdo (que captura atributos do usuário e item), para fornecer recomendações de projetos. Utilizamos a classificação tradicional de SRs, adaptada de Isinkaye et al. [54] para apresentarmos os três principais métodos, que descrevem a recomendação por meio da personalização.

## 2.2 Métodos de Classificação

No geral, os sistemas de recomendação se beneficiam dos métodos de classificação. Estes possibilitam a predição de quando um usuário irá aceitar certas recomendações de projetos ou não. Neste caso, podemos modelar o nosso problema como uma classificação binária (não fazer *fork* - 0 ou fazer *fork* - 1). Porém, a probabilidade do usuário aceitar ou não a recomendação é um fator que deve ser levado em consideração também em sistemas de recomendação. Essa informação pode ser utilizada para ranquear os itens quanto à probabilidade de a recomendação ser aceita ou não. O ranqueamento é realização de uma ordenação feita de acordo com o $score$ estimado pela uma aplicação. A partir do *score* pode ser construída uma lista de recomendações para ser apresentada para o usuário. Assim, é necessário que nosso classificador retorne uma lista ranqueada (um *score*), onde os maiores pontuadores têm maior probabilidade de serem recomendados e aceitos pelo usuário.

Modelos de predição são usados com sucesso em outros estudos para inferir, por exemplo, a popularidade do conteúdo em outras redes sociais, como o número de visualizações de vídeos do YouTube [82] [37] e o número de tweets associados a uma determinada hashtag [107] [71]. No entanto, para o nosso conhecimento, somos os primeiros a tentar prever a realização de *fork* de projetos de software hospedados no GitHub. A seguir, descreveremos como ocorre a classificação e quais são as abordagens utilizadas neste estudo para conhecermos sobre os classificadores supervisionados e não supervisionados.

### 2.2.1 Filtragem Baseada em Conteúdo

Escolhemos utilizar a Filtragem Baseada em Conteúdo por ser uma das técnicas de sistemas de recomendação mais populares e amplamente implementadas [1] [70]. Os métodos de filtragem baseada em conteúdo (FBC) filtram itens com base na semelhança do conteúdo em que o usuário está interessado. Essas abordagens utilizam características diferentes, como informações sobre o perfil do item ou do usuário, para aprender os fatores latentes de usuários ou itens. Dessa maneira, mesmo para um usuário que forneceu poucas interações, sua preferência ainda pode ser de alguma forma inferida. Por exemplo, um novo usuário no sistema de recomendação de restaurantes pode receber recomendações relevantes apenas considerando a cidade em que vive e definindo em seu perfil (por exemplo, vegetariano).

Outra vantagem dos métodos de filtragem baseada em conteúdo é que eles fornecem transparência sobre o funcionamento do sistema de recomendação. Assim, o usuário pode entender as recomendações fornecidas. Estudo realizado por Pu and Chen [84] com alguns usuários indicou que as interfaces explicativas poderiam efetivamente ajudar a criar um ambiente de confiança dos usuários no sistema. Outros pesquisadores também afirmam que as interfaces explicativas podem cultivar a confiança do usuário [98], promover a lealdade, aumentar a satisfação e tornar mais fácil para os usuários a busca pelo o que desejam encontrar [77]. Entretanto, uma limitação importante do sistema de recomendação baseado em conteúdo é que ele recomenda os mesmos tipos de itens - conhecido como efeito bolha - e por isso sofre de um problema de superespecialização [34].

Um método popular da FBC para recomendações baseadas em texto vem da *Information Retrieval*. *Term Frequency-Inverse Document Frequency* (TF-IDF) [85] é um método estatístico capaz de identificar as palavras mais relevantes de um documento, após ser treinado em um conjunto de documentos relacionados. Ele cria uma representação vetorial esparsa dos documentos, que pode ser usada para calcular a semelhança entre eles (por exemplo, usar a similaridade do Cosseno). Quando aplicado para recomendação, os vetores de perfis de usuário são agregados a partir dos vetores TF-IDF do item, utilizando a média ponderada, e os itens mais semelhantes são recomendados aos usuários. Em um de nossos trabalhos anteriores, o TF-IDF foi usado para recomendações de modelos UML, onde os alunos escolhiam quais características gostariam de encontrar nos modelos UML de diagramas de sequência [22].

Os sistemas de recomendação baseados em conteúdo recomendam itens com características semelhantes aos de um determinado usuário no passado. A similaridade dos itens é calculada com base nos recursos associados aos itens comparados, utilizando-se também de um vetor de palavras-chave. Por exemplo, se um usuário classificou positivamente um filme que pertence ao gênero de comédia por meio do feedback explícito, o sistema pode aprender a recomendar outros filmes desse gênero. Como a maioria dos recursos do GitHub não tem feedback explícito, também recuperamos as preferências do usuário a partir de seu feedback implícito, que reflete indiretamente o comportamento do usuário [51] [78].

### 2.2.2 Filtragem Colaborativa (FC)

A Filtragem Colaborativa é uma família de métodos bem conhecida para os SRs. A principal ideia por trás dos métodos de FC é que os usuários que compartilharam interesses semelhantes no passado tendem a compartilhar interesses semelhantes no futuro. Os algoritmos de filtragem colaborativa geralmente dependem dos dados históricos dos usuários. Por exemplo, suponha que um usuário chamado Pedro comprou livros que também foram comprados por Paulo. Algoritmos de filtragem colaborativa acabariam por inferir que Paulo tem preferências semelhantes às preferências de Pedro e, portanto, os outros livros comprados por Paulo podem ser considerados de grande interesse para Pedro. A família FC pode ser dividida em dois grupos: (a) FC baseada em memória e (b) FC baseada em modelo [34].

#### FC Baseada em Memória

A FC baseada em memória foi uma das primeiras técnicas de recomendação baseada em computador. Este método tenta encontrar usuários semelhantes ao usuário alvo e usa suas preferências para prever classificações para este usuário. As classificações ou interações são usadas para calcular a semelhança e o peso entre usuários ou itens [16].

Existem muitas vantagens no uso da FC baseada em memória. Primeiramente, é preciso calcular a similaridade entre os elementos. Segundo, estes sistemas de FC baseada em memória são escaláveis para uma grande quantidade de dados. Terceiro, a maioria dos sistemas baseados em memória são modelos de aprendizagem online. Assim, os dados gerados continuamente podem ser facilmente manipulados. Por fim, os resultados da recomendação podem ser entendidos e fornecer feedback para explicar por que recomendar esses itens [119].

No entanto, também existem algumas limitações nas técnicas de FC baseadas em memória. A técnica mostra-se lenta, pois utiliza toda base de dados a cada realização de uma predição. Os resultados da recomendação nem sempre são confiáveis ou precisos quando são utilizados poucos dados [104] [1] [108].

A FC baseada em vizinhança é um tipo de algoritmo de recomendação baseado em memória e envolve duas etapas: o cálculo de similaridade e a predição. Na etapa do cálculo de similaridade, métricas de similaridade (por exemplo, similaridade de Pearson ou Coseno) podem ser utilizadas entre usuários (*User-kNN*) ou itens (*Item-kNN*). Para gerar recomendações

para um determinado usuário, um conjunto de usuários semelhantes (vizinhos) é selecionado e os itens mais relevantes desses usuários são recomendados [49].

### FC Baseada em Modelo

A FC baseada em modelo é baseada em aprendizado de máquina ou em modelos de mineração de dados e pode alavancar padrões complexos de classificação nos dados de treinamento. Os algoritmos de FC baseada em modelo são desenvolvidos para combater as deficiências dos modelos de FC baseado em memória [119].

Algumas das vantagens dos algoritmos de FC baseada em modelo é melhorar as predições e, geralmente, são escaláveis por serem uma versão compactada dos padrões de comportamento dos usuários. Entretanto, esses modelos também sofrem com o problema de esparsidade, não sendo possível gerar recomendações razoáveis quando se tem poucas informações de classificações fornecidas pelos usuários [11].

Alguns exemplos desses métodos incluem árvores de decisão, redes neurais, redes bayesianas e métodos de fatores latentes e outros. Inicialmente, foram utilizados modelos como Naive Bayes [75] e Bayesian Networks [95] na exploração de recomendações. Entretanto, a fatoração de matrizes (FM) [62] é a abordagem de filtragem colaborativa mais bem-sucedida. Por exemplo, pode-se usar fatoração de matriz para encontrar matrizes densas representando usuários e itens, cujo produto reconstrói a matriz de classificação original. O método encontra fatores comuns em um espaço latente no conjunto de recursos de usuário e item, ou seja, esses podem ser os motivos subjacentes às interações dos usuários. Por exemplo, em um sistema de recomendação de filmes, esses fatores podem ser gênero, atores, diretor ou efeitos especiais. Ao final, a FM encontra um conjunto de características para cada usuário e item, tendo como resultado final o processo de fatoração. Por fim, após o treinamento, a preferência do usuário por um item pode ser estimada pelo produto escalar entre os vetores de fatores desses usuários e itens [119].

### Árvore de Decisão

Árvores, de um modo geral em computação, são estruturas de dados formadas por um conjunto de elementos que armazenam informações conhecidos por nós. Além disso, toda árvore possui um nó chamado raiz, que possui o maior nível hierárquico (o ponto de partida) e liga-

ções para outros elementos, denominados filhos. Esses filhos podem possuir seus próprios filhos que por sua vez também possuem os seus filhos. O nó que não possui filho é conhecido como nó folha ou terminal. Tendo essas definições esclarecidas, uma árvore de decisão nada mais é que uma árvore que armazena regras em seus nós, e os nós folhas representam a decisão a ser tomada.

Em uma árvore de decisão, uma decisão é tomada através do caminhamento a partir do nó raiz até o nó folha. Ou seja, árvore de decisão são classificadores em um atributo (ou classe) de destino na forma de uma estrutura em árvore. Os itens a serem classificados contêm o valor de atributo e o atributo de destino. Os nós da árvore podem ser: i) nós de decisão têm um único valor de atributo que é testado para determinar a qual ramificação da subárvore se aplica; ou ii) nós de folha que indicam o valor do atributo de destino. Entretanto, uma única árvore não é forte o suficiente para gerar melhores resultados na predição. Geralmente, utiliza-se modelos compostos por conjuntos de árvores, que resume a previsão de várias árvores juntas em um resultado final e isso pode ser realizado com a colaboração da aprendizagem de máquina, onde podemos organizar melhor as árvores para tomada de decisão. No geral, utiliza-se algoritmos que possam representar as árvores de forma não-supervisionada ou supervisionada.

### Máquina de Fatoração

Modelos de fatoração, que incluem máquinas de fatoração, são modelos populares em estatística e aprendizado de máquina. A máquina de fatoração (MF) é um modelo supervisionado poderoso que amplia significativamente a fatoração da matriz. Ela é adequada para dados transacionais de cardinalidade muito alta e escassamente observados, como observado em algumas características do GitHub (por exemplo, *pull-requests*). Nos trabalhos indicados, o autor Rendle [89] [90] propôs máquinas de fatoração para sistemas de recomendação e predição de taxa de cliques.

Nesta seção damos uma breve descrição matemática da máquina de fatoração. Assumindo um conjunto de treinamento $X_{train} = (x_i, y_i)$, com $i = 1, \ldots, n$, onde $x_i$ refere-se à i-ésima $i - th$ característica e $y_i$ refere-se ao i-ésimo $i - th$ alvo, o modelo de máquina de

fatoração da ordem 2 é escrito como mostra a Equação 2.1

$$\hat{y}(x) = w_0 + \sum_{j=1}^{p} w_j.x_j + \sum_{j=1}^{p}\sum_{j'=1}^{p} x_j.x'_j \sum_{f=1}^{k} v_{jf}.v_{j'f} \tag{2.1}$$

onde $x = (x_1, \ldots, x_p)$ é um vetor de característica de entrada $p$-dimensional observado, é o alvo de predição, $w_0$ é um fatores globais, $w_j$ são os fatores por característica e $v_{ij}$ denota a função coordenada $f$ do fator $k$-dimensional do vetor $v_j$. A matriz fatorial geral $\mathbf{V}$ do tamanho $p \times k$ é a concatenação dos vetores de linha $v_j$ para $j = 1, \ldots, p$. O número de fatores é $k$ e os parâmetros do modelo a serem estimados são $w_0, w_1, \ldots, w_p$ e $\mathbf{V}$.

Como já citado, os sistemas de recomendação visam aprender as preferências do usuário para recomendar itens, no nosso caso são projetos. O objetivo é prever qual classificação um usuário poderia fornecer hipoteticamente a um conjunto de itens e, em seguida, recomendar itens que o usuário provavelmente tenha interesse. Conforme ilustrado na Figura 2.3, usuários e itens formam uma matriz. Essa matriz é potencialmente muito grande, porque pode haver milhões de usuários e itens. Além disso, é escassamente observada, porque geralmente apenas uma fração muito pequena das classificações históricas está disponível.

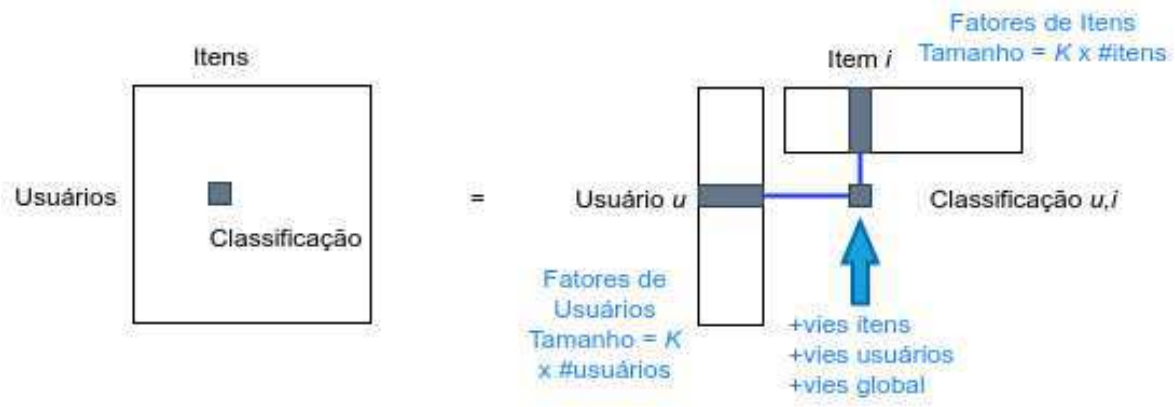


Figura 2.3: Fatoração matricial para um sistema de recomendação. Usuários e itens são caracterizados por seus respectivos vetores fatoriais k-dimensionais.

Esses desafios podem ser superados fatorando a matriz em fatores de usuário e item de menor dimensão, que podem ser usados para prever novas classificações. Para isso, construímos um vetor de entrada usando variáveis binárias para usuário $u$ e item $i$, como ilustrado na Figura 2.4.

Os dados da matriz podem ser facilmente transformados em vetores de recursos e *one-*

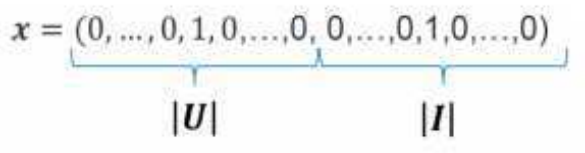


Figura 2.4: Vetor de entrada para um sistema de recomendação.

*hot encoding*. O modelo de máquina de fatoração é equivalente à Equação 2.2 para prever novas classificações:

$$\hat{y}(x) = \hat{y}(u, i) = w_0 + w_u + w_i + \sum_{f=1}^{k} v_{uj}.v_{if} \tag{2.2}$$

**Exemplos de Recomendadores Baseados em Estatísticas Simples**

Os SRs dependem de vários tipos de entrada. O mais comum é o feedback explícito, no qual os usuários informam diretamente o seu interesse por um item. Ou seja, um usuário pode classificar diferentes itens utilizando diferentes níveis de classificação. Um nível de classificação utilizado no contexto de "algo especial" é a quantidade de estrelas. Um usuário expressa explicitamente sua preferência por um livro sobre ficção científica, por exemplo, por meio de um conjunto $1, 2, 3, 4, 5$ de estrelas, onde pode classificá-lo com 5 estrelas, que representa a preferência máxima ou pode classificá-lo com 1 estrela, demonstrando sua preferência mínima pelo livro. As classificações mais comuns limitam-se em 5 estrelas e um pouco menos comum em 3 ou 4.

Nos casos em que o usuário não expressa explicitamente sua preferência por um item, uma estratégia alternativa é confiar e capturar os dados sobre o comportamento do usuário sem explicitamente solicitá-los, ou seja, capturar o feedback implícito [51] [78]. Por exemplo, histórico de compras, histórico de navegação, padrões de pesquisa ou até movimentos do mouse do computador são formas de capturar o feedback implícito.

Essas interações entre um usuário $u$ que pertence a um conjunto de usuários $U$ e o item $i$ que pertence ao um conjunto de itens $I$ são normalmente modeladas em uma matriz esparsa $M$. Formalmente, usuários e itens são representados por vetores que correspondem às linhas e colunas da matriz, onde $r_{(u,i)}$ representa a nota do usuário $u$ ao item $i$.

$$\vec{u} = (r_{(u,1)}, r_{(u,2)}, \ldots, r_{(u,|I|)})^T$$

$$\vec{i} = (r_{(1,i)}, r_{(2,i)}, \ldots, r_{(|U|,i)})$$

Na Tabela 2.1, alguns itens não foram classificados pelo usuário. Neste caso, o sistema de recomendação pode realizar a previsão da nota, como normalmente é feito na técnica de filtragem colaborativa, considerando as classificações dos usuários para os itens. Por exemplo, ter como preditor a média global das notas ou usar a variação das notas de usuários e itens individuais em relação à média global.

Tabela 2.1: Matriz Esparsa representando Feedback Explícito

| | $i_1$ | $i_2$ | $i_3$ |
|---|---|---|---|
| $u_1$ | 1 | ? | 3 |
| $u_2$ | 4 | ? | ? |
| $u_3$ | ? | 5 | 5 |

A média global das notas seria calculada pela Equação 2.3.

$$\mu = \frac{1}{n} \sum_{(u,i,r) \in D^{train}} r_{u,i} \tag{2.3}$$

Os conjuntos de treinamento ($D^{train}$) e de teste ($D^{test}$) foram criados. O conjunto de treinamento é usado para aprender os parâmetros ou configurar os algoritmos usados na etapa de análise dos dados para ajustes no algoritmo de recomendação, enquanto o conjunto de dados de teste é usado para avaliar o modelo ou a configuração obtida na fase de treinamento, garantindo um bom desempenho com dados não vistos anteriormente.

A partir da média global, podemos calcular a variação das notas, de acordo com a Equação 2.4

$$\hat{r}(u,i) = \mu + b_u + b_i \tag{2.4}$$

onde $b_u$ e $b_i$, respectivamente, são os desvios das notas de $u$ e $i$ em relação à média global.

$$b_u = \bar{r}_u - \mu$$

$$b_i = \bar{r}_i - \mu$$

e $\bar{r}_u$ e $\bar{r}_i$ são as médias das notas do usuário $u \in U$ e item $i \in I$, respectivamente.

Portanto, aplicando a Equação 2.4 aos dados da Tabela 2.1, temos a previsão de nota para os valores desconhecidos anteriormente.

$$\hat{r}(u_1, i_2) = 3,6 - 1,6 - 1,1 = 0,9$$

$$\hat{r}(u_2, i_2) = 3,6 - 1,6 - 1,1 = 0,9$$

$$\hat{r}(u_2, i_3) = 3,6 - 1,6 + 0,4 = 1,6$$

$$\hat{r}(u_3, i_1) = 3,6 + 1,4 - 1,6 = 3,4$$

Tabela 2.2: Matriz Esparsa representando Feedback Explícito

| | $i_1$ | $i_2$ | $i_3$ |
|---|---|---|---|
| $u_1$ | 1 | 0,9 | 3 |
| $u_2$ | 4 | 0,9 | 1,6 |
| $u_3$ | 3,4 | 5 | 5 |

A suposição básica da filtragem colaborativa é que dois usuários que tiveram interesses semelhantes no passado tendem a compartilhar os mesmos interesses no futuro. Portanto, considerando que $u$ e $v$ são usuários pertencentes ao conjunto $U$, e $i$ e $j$ pertecem ao conjunto $I$, a abordagem de filtragem colaborativa utiliza-se da classificação dos usuários $u$ e $v$ para analisar suas relações. A ideia principal é que a classificação de $u$ sobre um novo item $i$ seja semelhante à de outro usuário $v$, se $u$ e $v$ classificaram outros itens de maneira semelhante. Da mesma forma, é provável que classifiquem dois itens $i$ e $j$ de maneira semelhante, se outros usuários tiverem classificações semelhantes para esses dois itens.

Para filtragem colaborativa baseada no usuário-alvo $u$ e um item $i$, podemos calcular os $k$ vizinhos mais próximos de $u \in U$

$$V_u := \underset{v \in U \setminus \{u\}}{\overset{k}{\operatorname{argmax}}}\, s(\vec{u}, \vec{v}) \tag{2.5}$$

$$V_u := \underset{v \in U \setminus \{u\}}{\overset{k}{\operatorname{argmax}}}\, s(\vec{u}, \vec{v})$$

Para, então, calculamos a média das notas dos vizinhos para o item $i \in I$

$$\hat{r}(u,i) := \frac{1}{|V_u|} \sum_{v \in V_u} r_{v,i} \tag{2.6}$$

$$\hat{r}(u,i) := \frac{1}{|V_u|} \sum_{v \in V_u} r_{v,i}$$

Tomando o mesmo exemplo mostrado na Tabela 2.2, podemos aplicar as Equações 2.5 e 2.6, para encontrarmos o valor para $\hat{r}(u_2, i_3)$. Assumindo que $k = 2$ e $V_{u_1} = \{u_1, u_3\}$,

$$\hat{r}(u_2, i_3) = \frac{1}{2}(3 + 5) = 4$$

.

Resumindo, as técnicas de filtragem colaborativa contam com a similaridade usuário-usuário ou item-item para realizar a recomendação. Para isso, um sistema de filtragem colaborativa precisa definir duas entidades: itens e usuários. Cada usuário $u$ é associado a um subconjunto $I_u \subset I$ de itens avaliados por $u$. Da mesma forma, cada item $i$ é associado a um subconjunto $U_i \subset U$ de usuários que avaliaram o item $i$. As avaliações dos usuários são armazenadas na matriz $M$ onde cada elemento $m_{ui}$ representa o valor da avaliação (*rating*: 1-5, onde 1 é ruim e 5 é excelente) dada pelo usuário $u$ ao item $i$. Desta forma, um SR pode ter o objetivo de realizar a **predição** de um valor $p_{ui}$ que mede o grau de preferência do usuário $u$ pelo item $i$ ou **recomendar** uma lista de $N$ itens recomendados, que o usuário $u$ tenha preferência, ordenados de acordo com o $p_{ui}$ estimado pela aplicação, exibindo $Top - N$ itens.

### 2.2.3 Filtragem Híbrida (FH)

Os sistemas híbridos de recomendação surgiram à medida que várias estratégias de recomendação amadureceram, combinando dois ou mais algoritmos em sistemas compostos, que se baseiam idealmente nos pontos fortes de seus algoritmos de componentes para obter alguma sinergia. Uma pesquisa realizada por Burke [19] identificou sete estratégias diferentes para o RS híbrido: ponderado, comutação, misto, combinação de características, incremento de características, cascata e meta-nível. Seis desses tipos híbridos foram explorados, combinando quatro métodos básicos de recomendação: filtragem baseada em conteúdo, filtragem

colaborativa, colaboração heurística e filtragem baseada em conhecimento, gerando 41 combinações possíveis de técnicas híbridas [20]. Um dos trabalhos pioneiros que exploraram a combinação híbrida de revisões de informações textuais com a classificação dos usuários foi do autor Jakob et al. [56]. Ele descobriu que as críticas de filmes geralmente estão relacionadas a diferentes aspectos, como preço, serviço, sentimentos positivos ou negativos, e os aspectos que podem ser explorados para a previsão de classificação.

## 2.3 Redes Sociais

O conceito de redes sociais foi introduzido pela primeira vez por Barnes and Harary [8] que a descreve como um grafo $(N, r)$, onde $N$ é o conjunto de nós (ou vértices) que representa os indivíduos (pessoas, organizações, países, etc.), e $r$ é o conjunto de arestas (interações, convites, rastreamento, valores ou *links*) representando seus relacionamentos, dada por uma matriz $n \times n$, no qual $r_{i,j}$ é a relação (peso ou direção) entre os nós $i$ e $j$. Como exemplo, a Figura 2.5 mostra uma rede social entre amigos representada por um grafo, onde os nós são as pessoas e os *links* representam a amizade entre essas pessoas.

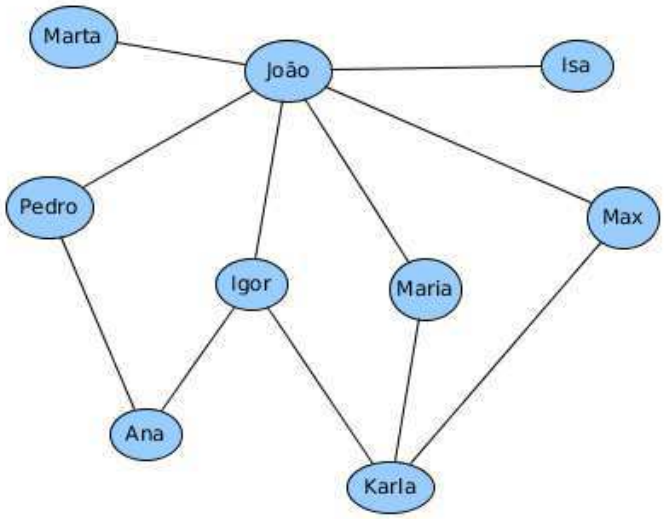


Figura 2.5: Exemplo de uma rede social de amizade

O surgimento de Redes Sociais On-Line (RSO) mudou o estilo de vida das pessoas, pois ampliou a noção de redes sociais em termos de tamanho [42] e acessibilidade [76], tornando-

as presentes em todos os lugares [53].

Inicialmente, o uso dessas redes sociais era limitado à conexão entre amigos e familiares [48]. Exemplos típicos incluem as plataformas de RSO Facebook e YouTube, que permitem o compartilhamento de informações e alavanca as relações entre os usuários das plataformas [4]. Além disso, o estudo de Travers and Milgram [106] revela a alta probabilidade de dois indivíduos serem amigos se estes tiverem amigos em comum.

O crescimento fenomenal das RSO não passou despercebido pelas empresas e governos, que começaram a explorar esse potencial uso das redes sociais para fornecer e melhorar os serviços prestados [55]. O uso das RSO com SRs criaram novas oportunidades para empresas que consideram a influência social relevante para o marketing do produto, bem como as redes sociais que pretendem melhorar a experiência do usuário personalizando o conteúdo que é fornecido a cada indivíduo e permitindo novas conexões. Do mesmo modo, essas mudanças criaram novos desafios para os pesquisadores na área de SRs e análise de redes sociais [35].

Novas oportunidades surgem para geração de novos algoritmos, por exemplo, para analisar conexões dentro da estrutura gráfica da rede social, fazer recomendações sobre conexões indiretas e compreender a estrutura de relacionamentos e colaboração, melhorando-os com o passar do tempo [26], [116], [120]. Portanto, à medida que a lista de possíveis recomendações cresce, a importância de algoritmos que retornem recomendações mais precisas e utilizem dados mais refinados também cresce [66].

Existem várias plataformas de redes sociais para desenvolvedores, onde os usuários (desenvolvedores de software) se inscrevem em uma conta, interagem e compartilham informações com outros usuários (por exemplo, contribuir em projetos ou repositórios de software, e responder a perguntas adicionadas nas plataformas). O Github é uma das plataformas de redes sociais mais populares, onde os usuários podem pesquisar por grandes quantidades de código, distribuir códigos de outros usuários e criar ramificações para projetos [52] [109].

Neste trabalho, utilizamos a rede social GitHub para propormos um modelo de recomendação de projetos de software para os usuários. A recomendação de projetos no GitHub tem por objetivo estimular a colaboração entre os usuários da plataforma e, para isso, consideramos os principais aspectos que influenciam nestas relações de colaboração para alcançarmos recomendações relevantes, como por exemplo, as atividades do usuário na plataforma (seguir um usuário ou observar um projeto) ou as linguagens de programação de projetos de

softwares mais utilizadas na plataforma.

## 2.4 GitHub

Atualmente, o GitHub é a principal plataforma de codificação social utilizada pelos desenvolvedores e contém uma premissa bem simples: fornecer uma solução de alto nível que permita aos desenvolvedores criar e manter projetos de software de código aberto utilizando um sistema de controle de versão distribuído chamado Git. Diferentemente do sistema de versão Subversion[1], no Git não existe uma cópia principal de um projeto. Todas as cópias são documentos de trabalho, onde os desenvolvedores podem realizar mudanças locais em uma cópia de trabalho sem precisar estar conectado a um servidor central.

A capacidade dos desenvolvedores de criarem e colaborarem em projetos de software de maneira simples e com esforço mínimo (uma vez que se entenda alguns detalhes fundamentais sobre como funciona a ferramenta Git), certamente simplificou muitos dos detalhes tediosos que poderiam impedir a colaboração. A plataforma GitHub é um facilitador do desenvolvimento de software de código aberto, permitindo a inovação, facilitando a criação de projetos, compartilhamento de código-fonte e, tudo isso, realizado de forma transparente.

A Figura 2.6 é um exemplo do ambiente gráfico de desenvolvimento de um usuário do GitHub. Este painel pessoal principal exibe as atividades dos usuários no GitHub. Por meio dele, o usuário pode acompanhar os problemas e receber notificações sobre os projetos que observa ou realiza atualizações. Também é possível navegar pelos principais repositórios e visualizar o conteúdo de algum projeto específico, identificando suas atividades mais recentes e a qual organização ou usuário pertence o repositório.

Ainda neste ambiente, o usuário pode hospedar e revisar código, gerenciar seus projetos e criar software colaborativamente com outras centenas de milhares de desenvolvedores, ter seus projetos observados (*watchers*) ou observar os projetos de de seu interesse e, ter seguidores (*followers*) e seguir outros usuários, copiar projetos de seu interesse para seu ambiente local (*fork*), recomendar melhorias de código-fonte nos projetos públicos (*pull-request*) e outras funcionalidades.

O GitHub suporta dois modelos de desenvolvimento colaborativo [43]: (1) o modelo de

[1] `https://subversion.apache.org/`

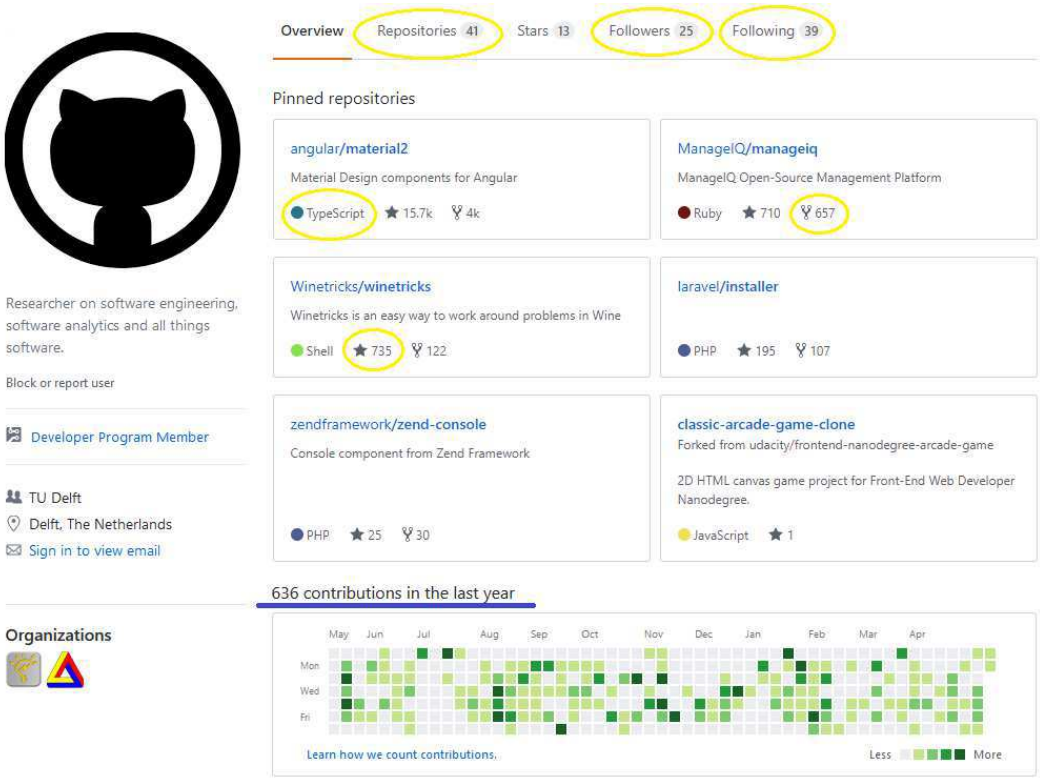


Figura 2.6: Exemplo do ambiente GitHub - Perfil

repositório compartilhado e (2) o modelo de desenvolvimento baseado em *pull*. O primeiro concede acesso aos usuários que realizam alterações diretamente neste repositório compartilhado. Toda mudança é publicada imediatamente e não passa por um processo de revisão. O uso deste modelo é frequentemente utilizado por pequenas equipes e organizações que colaboram em projetos privados.

Atualmente, quase metade dos projetos no GitHub usam o modelo de desenvolvimento baseado em *pull* [44]. Ele contrasta com o modelo de repositório compartilhado por existirem membros da equipe principal do projeto (conhecidos por integradores) que são responsáveis por avaliar a qualidade das contribuições, propondo correções, conversando com os contribuidores (por meio de *issues*) e, eventualmente, mesclando ou rejeitando as mudanças. Este modelo visa dar transparência às discussões abertas em torno das solicitações de *pull-request* e torna o processo global mais democrático. Os conceitos mais relevantes e aplicados na plataforma GitHub estão descritos na Tabela 2.3.

Particularmente, os projetos envolvidos no modelo de desenvolvimento baseado em *pull*

tendem a ser maiores e, por serem públicos, permitem que estudos possam ser realizados com base em seus aspectos e implicações (por exemplo, Yu et al. [114] analisam os fatores que influenciam na avaliação de uma solicitação de *pull-request* e Gousios et al. [44] examinam os aspectos relacionados às práticas e desafios do trabalho de desenvolvimento baseado em *pull*).

A Figura 2.7 ilustra, de forma simplificada, o modelo baseado em *pull*. Um usuário do GitHub pode criar seus próprios repositórios e modificá-los. Caso ele tenha interesse em utilizar um *framework* em seu projeto, por exemplo o Hibernate [9], precirá fazer um *fork* deste repositório. O *fork* é uma maneira de duplicar o repositório, obtendo a sua versão mais atual, e permite que o usuário trabalhe de forma independente e realize alterações apenas na versão que encontra-se no seu próprio ambiente de trabalho. Se desejar, ele pode submeter as alterações para o repositório original por meio de um *pull-request*.

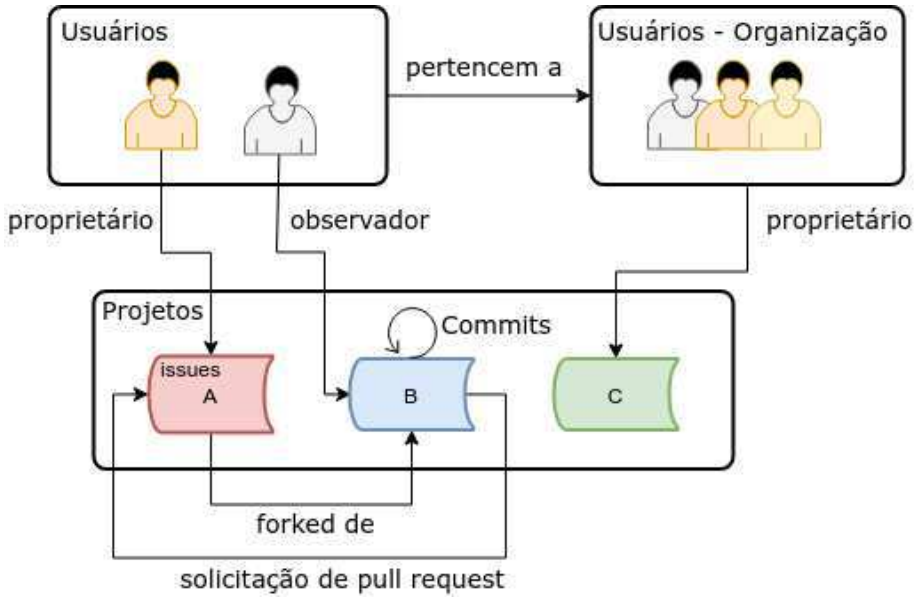


Figura 2.7: Simplificação do Modelo Baseado em *Pull*

O desenvolvedor central do repositório revisa as alterações submetidas via *pull-request* e decide se aceita ou não tais alterações. Muitas discussões, conhecidas por *issues* podem ocorrer durante o processo de aprovação do *pull-request* e essas discussões podem ser abertas para discussão de questões relevantes. Além das discussões via *issues*, pode-se acessar o Gitter[2], um serviço relacionado e que facilita as discussões entre os membros da comunidade

[2]http://gitter.im/

do GitHub, fornecendo um bate-papo integrado para solução de problemas. Caso o novo código submetido no *pull-request* seja aceito, o autor torna-se um dos contribuidores do projeto.

Portanto, as atividades dos modelos de desenvolvimento social resultam na geração de características contextuais estáticas (características extraídas em um determinado momento, como por exemplo, quantidade de observadores) e dinâmicas (características que evoluem na linha de vida do usuário, como por exemplo, a atividade do projeto) [60] que podem ser extraídas para uma análise mais abrangente e revelar recursos que possam contribuir para as predições e recomendações deste estudo.

## 2.5 Considerações Finais

Neste capítulo, apresentamos a fundamentação sobre sistemas de recomedação por meio da discussão dos conceitos de seus principais elementos, abordagens, modelos e estratégias que têm sido propostas para utilizar sistemas de recomendação em diversas áreas de negócio e como melhorar o desempenho desses sistemas. Neste trabalho, buscamos entender como recomendadores podem ser utilizados pela rede social GitHub, que é um ecossistema onde comunidades de projetos podem depender uns dos outros, incentivando a colaboração entre usuários que utilizam uma plataforma transparente, construída por meio de características sociais. Há uma ampla literatura de conceitos e teorias desenvolvidas nessas áreas que servem de base teórica para o estudo conduzido neste trabalho. Essa literatura é apresentada nos próximos capítulos, quando se detalha os estudos realizados na contextualização do ecossistema GitHub e nas recomendações de projetos.

Tabela 2.3: Conceitos do GitHub

| **Conceito** | **Descrição** |
|---|---|
| *Colaborator* (Colaborador) | Pessoa com acesso de leitura e gravação a um repositório que foi convidado a contribuir pelo proprietário do repositório. |
| *Commit* (Confirmação) | Um *commit* ou "revisão" é uma alteração individual em um arquivo (ou conjunto de arquivos). As confirmações geralmente contêm uma mensagem de confirmação, que é uma breve descrição de quais alterações foram feitas. |
| *Follower* (Seguidor) | É permitido seguir pessoas no GitHub para receber notificações sobre suas atividades e descobrir projetos em suas comunidades. |
| *Fork* (Cópia) | Nome dado à ação de criar um novo repositório a partir da cópia de um outro repositório existente. |
| *Issue* (Assunto) | Sugestões de melhorias, tarefas ou perguntas relacionadas ao repositório. Cada projeto contém seu próprio fórum de discussão, podendo a *issue* ser rotulada e visualizada por qualquer usuário. |
| *Organizations* (Organizações) | As organizações são contas compartilhadas nas quais empresas e projetos de código aberto podem colaborar em vários projetos ao mesmo tempo. Proprietários e administradores podem gerenciar o acesso dos membros aos dados e projetos da organização com sofisticados recursos administrativos e de segurança. |
| *Pull-request* (Requisição) | Representa a atualização em um repositório, quando realizado por usuário não proprietário do projeto. Para isso, é preciso que o repositório tenha sido clonado e o usuário tenha solicitado ao proprietário do repositório a requisição de atualização na plataforma. |
| *Watcher* (Observador) | Qualquer usuário que tenha por intenção observar as atualizações de um repositório, podendo receber notificações a respeito. |

# Capítulo 3

# Trabalhos Relacionados

Neste capítulo, apresentamos os esforços de estudos anteriores no domínio de recomendações de projetos no GitHub. Iniciamos apresentando alguns estudos relacionados ao ecossistema GitHub (Seção 3.1), juntamente com os resultados do *Open Source Survey*[1], que é um projeto de dados abertos do GitHub e de colaboradores da academia, indústria e comunidade de código aberto. Finalizamos o capítulo comparando os trabalhos de SRs anteriores com a nossa proposta de trabalho (Seção 3.2).

## 3.1 Ecossistema GitHub

Diversas estratégias têm sido propostas com o objetivo de caracterizar o GitHub, tanto como rede social quanto como plataforma colaborativa. Estudos com esse propósito têm considerado as distribuições do número de contribuidores por repositório, observadores por projeto, seguidores por usuário, laços sociais de colaboração mediada por repositórios, atividade dos projetos em termo de eventos criados, geografia dos centros de atividade, influência na colaboração e outros. Lima et al. [67] foi o primeiro estudo quantitativo sobre as interações entre os usuários do GitHub, destacando a reciprocidade baixa das interações entre eles e a colaboração com maior frequência em projetos pequenos, onde os usuários muito ativos na plataforma não têm necessariamente um grande número de seguidores.

Por ser uma plataforma colaborativa, o ambiente GitHub propicia o surgimento de ecossistemas de projetos, definidos como "uma coleção de projetos de software que são desenvol-

[1] http://opensourcesurvey.org

vidos e que evoluem em conjunto no mesmo ambiente", tornando-se uma área de interesse em pesquisas recentes [13]. Casalnuovo et al. [21] mostrou que os desenvolvedores optam por participar de projetos os quais eles tenham relações anteriores de trabalho. Entretanto, por si só, isso pode não ser suficiente para levar às contribuições contínuas a longo prazo.

Para análise do ecossistema GitHub, apresentamos a seguir dados de uma pesquisa executada pelo próprio GitHub - *Open Source Survey* - realizada em Março de 2017 (de acordo com o estudo de Geiger [39]). O GitHub projetou essa pesquisa com o objetivo de reunir dados sobre práticas e comunidades de desenvolvimento de software de código aberto. Os resultados são um conjunto de dados que demonstram as atitudes e experiências dos usuários que usam, criam e mantêm projetos de software de código aberto. Esta pesquisa apresenta as principais contribuições dos usuários no GitHub, as atividades que os usuários frequentemente contribuem, que é para codificação social e, ocasionalmente, sobre reportação de *bugs* ou ideias de novas funcionalidades, como vemos na Figura 3.1. A maioria dos contribuidores não se envolvem na execução de funções organizacionais ou administrativas (por exemplo, gerenciar listas de discussão, organizar eventos ou outros).

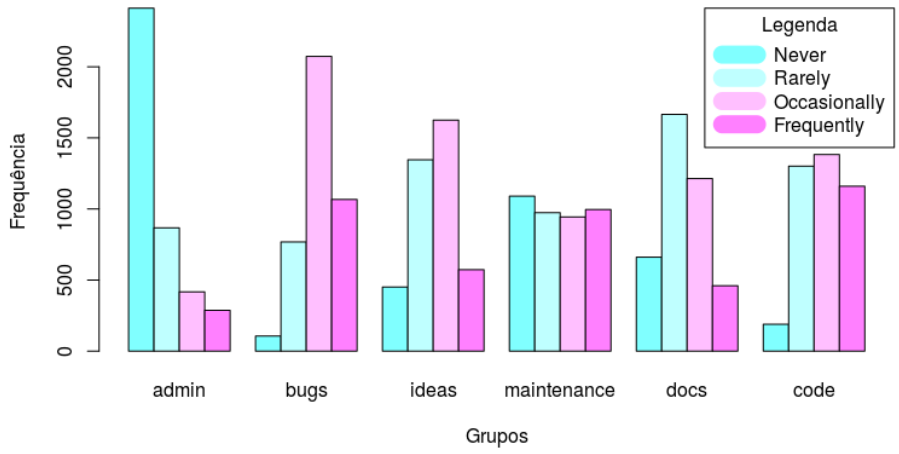


Figura 3.1: Principais Contribuições no GitHub (imagem apresentada na pesquisa de Geiger [39])

Uma das dificuldades mais evidentes identificadas na pesquisa para o processo de codificação social, de acordo com a Figura 3.2, é a documentação incompleta ou de difícil entendimento, sendo um problema generalizado e observado por 93% dos entrevistados. A documentação orienta os contribuidores menos experientes, por exemplo, informando como

usar ou contribuir em um projeto, ou quais são os termos de uso e os padrões de conduta da comunidade. Melhorar essa documentação é uma maneira importante de estimular os usuários para codificação social, entretanto, podemos perceber que a documentação acaba por ser um impeditivo para que os usuários possam contribuir com maior frequência.

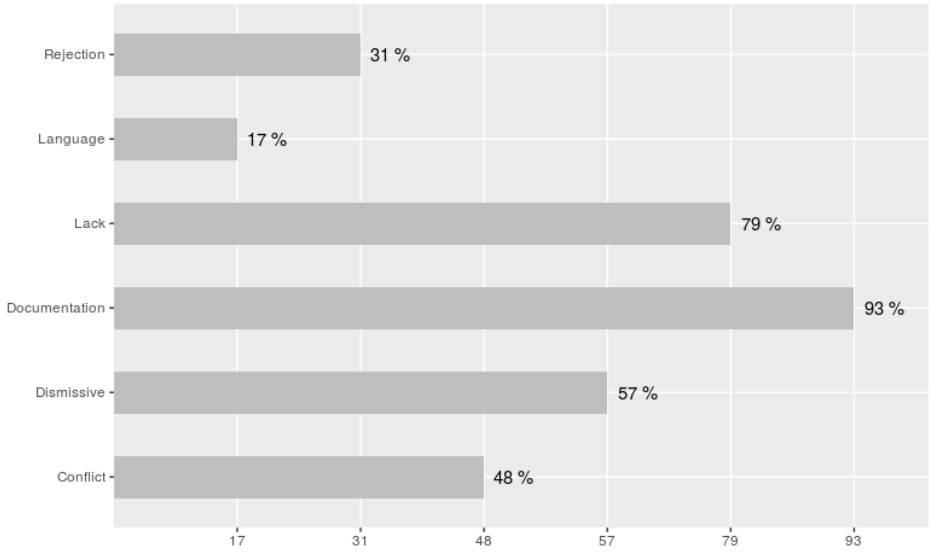


Figura 3.2: Dificuldades Identificadas no Processo *Open Source* (imagem apresentada na pesquisa de Geiger [39])

Então, o que motiva tantos desenvolvedores de sofware a gastarem tempo, habilidades e conhecimento neste tipo de ambiente? Os estudos [88] [99] [112] buscaram compreender a motivação, influência, atração, integração e retenção dos usuários recém-chegados e demais usuários nos projetos. Os estudos chegaram à conclusão que a aprendizagem é uma das principais forças motrizes para motivar as pessoas a se envolverem nestas comunidades. Outros pontos a serem destacados são os benefícios intrínsecos (por exemplo, diversão e identificação com uma comunidade), extrínsecos (melhores empregos, progressão na carreira, reputação) e os desafios derivados (escrever um novo código e melhorar as habilidades de programação). Entretanto, envolver os recém-chegados nos ecossistemas de projetos de forma significativa e estimular os usuários presentes na plataforma a conhecer ainda mais projetos dentro da plataforma é um grande desafio [100].

Ainda na Figura 3.2, a falta de resposta às contribuições ou perguntas feitas pelos contribuidores está entre os 79% dos problemas encontrados pelos respondentes. Outros dois problemas que chamam a atenção são respostas discriminatórias às contribuições ou perguntas e o conflito ou tensão interpessoal entre contribuintes, que estão com 57% e 48%,

respectivamente. Entretanto, mesmo com as dificuldades apresentadas, o número de contribuições aumentaram 25,1% em relação ao período anterior, como mostra a Figura 3.6.

A capacidade de observar projetos no GitHub introduziu uma nova camada de participantes de desenvolvimento de software, onde os usuários podem observar os projetos, receber notificações de alterações no código-fonte, mas não necessariamente precisam contribuir com os projetos [96]. Os estudos de Blincoe et al. [14] e Lee et al. [64] perceberam que os usuários que possuem uma certa influência, acabam por orientarem seus seguidores para novos projetos. Os autores mencionados sugerem a criação de novas ferramentas que permitam que os recém-chegados possam encontrar esses usuários influentes e, assim, serem motivados a contribuir e não abandonar a plataforma.

Na pesquisa de Geiger [39], quando perguntado sobre o interesse dos usuários em contribuir em projetos de código aberto no futuro e quão provável eles contribuiriam com esses projetos, vemos que os usuários estão muito interessados em fazer contribuições futuras, de acordo com o gráfico de barras esquerdo, mas estão menos propensos a dizer que são muito prováveis de realizar tais contribuições, como mostra o gráfico de barras direito na Figura 3.3. Entretanto, o importante neste ponto é que os usuários gostariam de contribuir mais futuramente.

Dabbish et al. [32] fornece informações sobre como a plataforma GitHub permite novos tipos de colaboração devido à transparência dos projetos para usuários na plataforma. Eles discutem a "utilidade colaborativa" e o valor quando "a transparência é integrada a um espaço de trabalho baseado na Web", bem como o valor das características sociais que tornam a atividade visível para os usuários, onde o GitHub ajuda no aprendizado de "melhores maneiras de codificar e acessar um conhecimento superior".

Portanto, a transparência dos projetos permite que os mesmos estejam disponíveis para pesquisa e reutilização de código [32]. Gharehyazie et al. [40] e Vasilescu et al. [109] perceberam que a codificação é mais rápida por meio da clonagem de projetos (*fork*), quebrando as fronteiras geográficas. Esses estudos encontraram evidências de dependências técnicas entre projetos, identificando a existência de padrões de código que podem ser candidatos ao compartilhamento de código e geração de ferramentas de recomendação [38] [94]. E, o meio principal de comunicação para pedir ajuda ou ajudar os usuários em seus projetos são os fóruns de discussão e, a segunda forma mais usual é falar diretamente com o usuário que

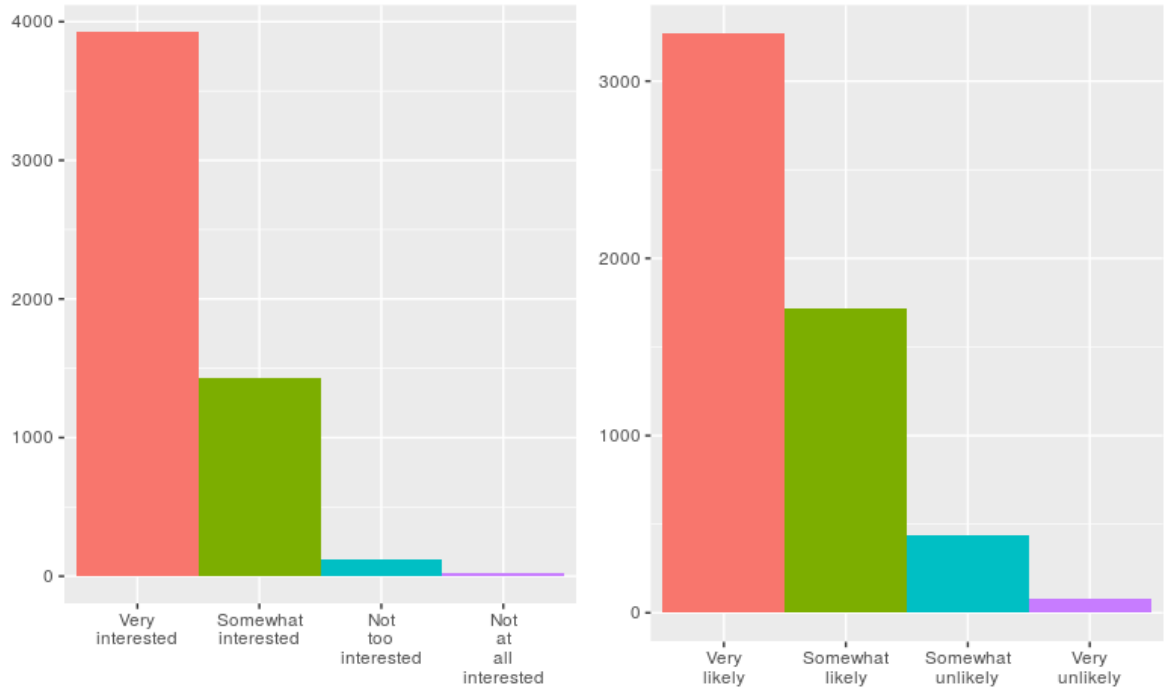


Figura 3.3: Usuários estão muito interessados em realizar contribuições futuras (lado esquerdo). Entretanto, não estão muito propensos a realizar tais contribuições (lado direito). (imagem apresentada na pesquisa de Geiger [39])

está necessitando de ajuda, como mostra a Figura 3.4, retirada do estudo de Geiger [39].

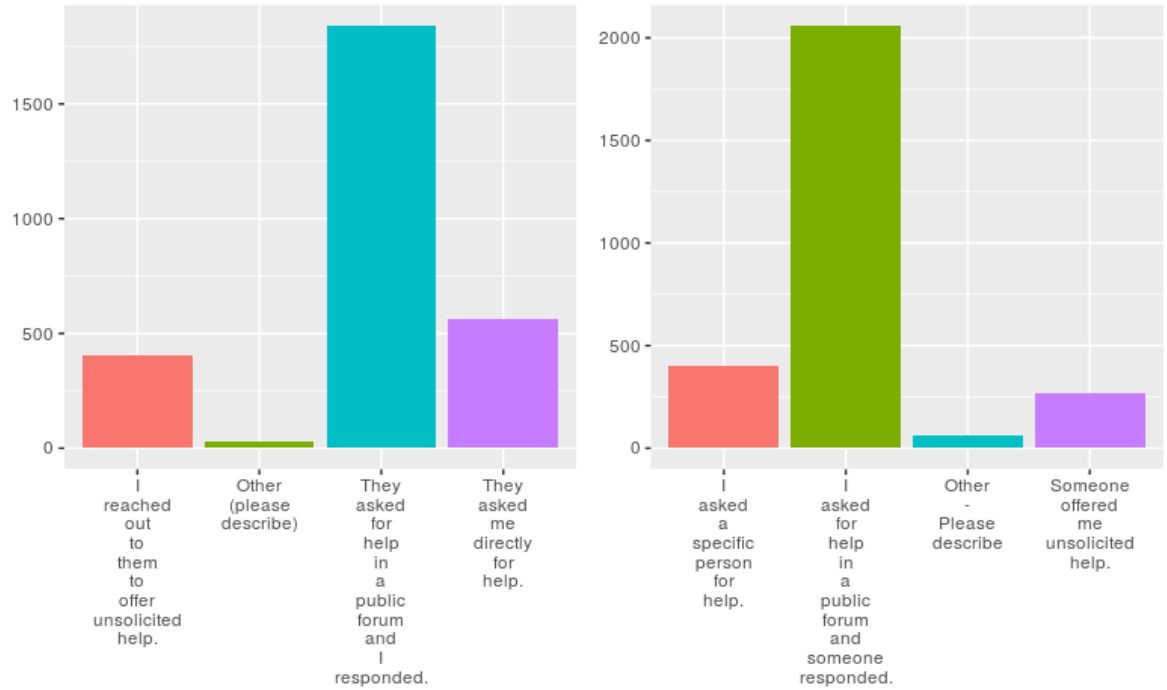


Figura 3.4: Usuários utilizam como principal meio de comunicação os fóruns, tanto para pedir ajuda (lado esquerdo) como para ajudar (lado direito). (imagem apresentada na pesquisa de Geiger [39])

Algumas pesquisas mostram que os usuários em redes sociais sentem-se mais à vontade em pedir ajuda ou falar com pessoas que sejam da sua convivência ou que tenham amizade com seu ciclo de amigos, mas os contribuidores do GitHub têm se mostrado diferentes [86] [97]. A Figura 3.5 deixa bem claro que a maioria dos contribuidores não tinham um relacionamento prévio com os usuários os quais estavam ajudando ou sendo ajudados, eram pessoas totalmente estranhas ao seu convívio e, mesmo assim, isso não foi impeditivo para que a contribuição ocorresse.

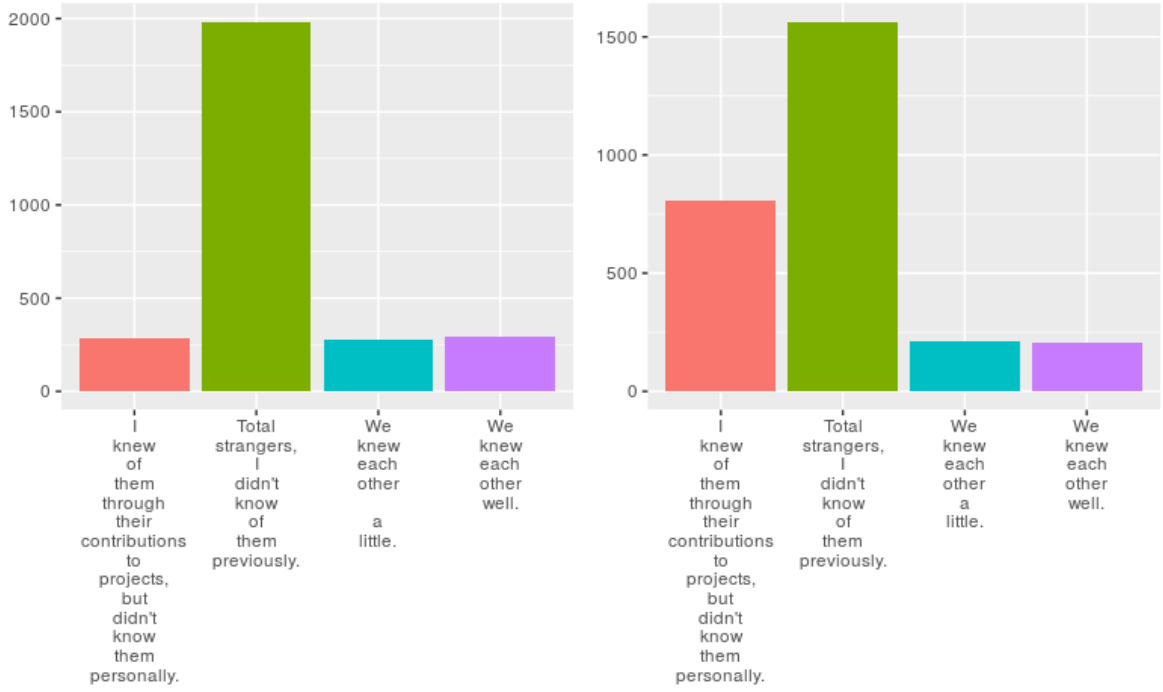


Figura 3.5: A maioria dos contribuidores não tinham um relacionamento prévio com os usuários os quais estavam ajudando (lado esquerdo) ou sendo ajudados (lado direito) (imagem apresentada na pesquisa de Geiger [39])

Mas, ainda assim, dentre os 83 milhões de repositórios no ano de 2018, apenas 19,7% dos repositórios são *fork* de outros projetos. Ou seja, a contribuição social tem muito a crescer ainda no GitHub e o uso de sistemas de recomendação podem ser de grande valor, no sentido de promover e aumentar a colaboração na plataforma. Os SRs podem ser utilizados para apresentar ao usuário alvo informações que favoreçam o seu interesse de contribuição com base nos dados de outros usuários com interesses semelhantes. Utilizar a informação de *fork* pode ser um alvo a ser utilizado pelos recomendadores baseado nas características dos projetos, ajudando os colaboradores a encontrarem projetos que estão abertos à contribuições.

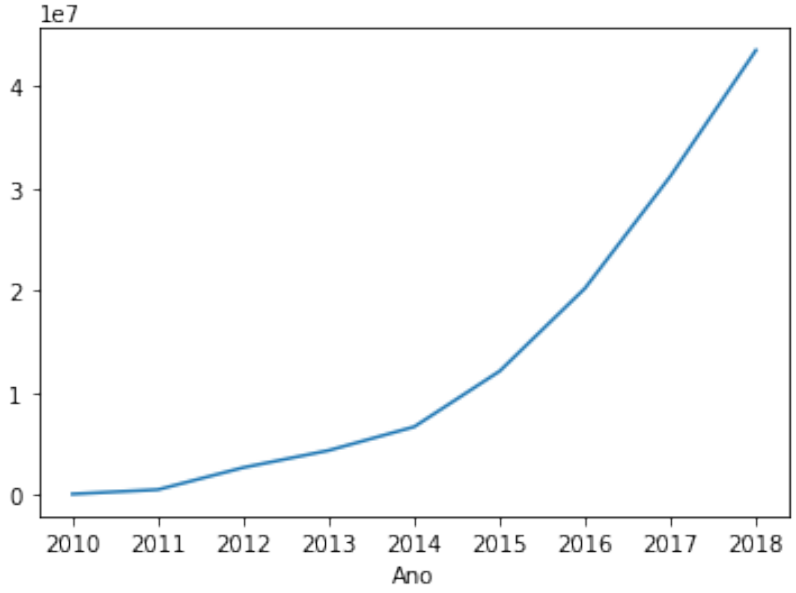


Figura 3.6: Quantidade de *Pull-Requests* por Ano

A disponibilização dos dados do GitHub motivou muitos estudos com o objetivo de caracterizar este novo contexto de colaboração. [32] [105] [59] [67] analisam os dados coletados do GitHub com o objetivo de entender melhor as interações e particularidades decorrentes deste ambiente, que combina características sociais e colaborativas. Embora não ofereça técnicas de recomendação, essas pesquisas produzem conhecimento sobre as motivações dos usuários do GitHub e nos ajuda a qualificar, motivar e explicar os resultados obtidos em nosso próprio trabalho.

## 3.2 Aplicações de Recomendação no GitHub

A Tabela 3.1 resume os trabalhos mais representativos sobre recomendação de sistemas no GitHub. Podemos perceber que algumas fontes de dados (*language*, *watchers*, *followers*, *pull-requests*, *fork*, *issues*, *commits*, *text*) foram usadas de forma semelhante entre alguns projetos. De acordo com os trabalhos listados, pode-se perceber que as características dos usuários e seus respectivos projetos, como também as relações entre os usuários da plataforma, podem abranger diferentes escopos em pesquisas científicas e desenvolvimento de sistemas de recomendação. Neste trabalho, apresentamos o conjunto de características de *visibilidade*, *atratividade*, *medidas estatísticas* e *engajamento*, que serão detalhadas no Ca-

pítulo 5.

Muitos dos estudos foram conduzidos com o propósito de entender o comportamento dos usuários em uma rede social colaborativa e alguns sugerem o uso de sistemas de recomendação de projetos, revisores de *pull-request*, focados, principalmente, em delinear os fatores que explicam o compartilhamento de projetos e conhecimento.

Tabela 3.1: Estudos sobre Sistemas de Recomendação no GitHub

| | **Fonte de Dados** | | | | | | | | | **Técnicas Empregadas** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Trabalhos** | Language | Watchers | Followers | Pull-Requests | Fork | Issues | Commits | Text | Engagement | Filtragem Colaborativa | Baseada em Conteúdo | Fatoração de Matriz | Máquina de Fatoração | Predição de Link | Rede Neural | Árvore de Decisão |
| Jiang et al. [58] | ✓ | ✓ | | | | | ✓ | | | ✓ | | | | | | |
| Yu et al. [115] | | | | ✓ | | ✓ | ✓ | | | ✓ | | | | | | |
| Orii [79] | ✓ | | | | | | | ✓ | | | | ✓ | | | | |
| Lee and Chen [65] | | | | ✓ | ✓ | ✓ | | | | ✓ | | | | ✓ | | |
| Liu et al. [68] | ✓ | | | | | | ✓ | | ✓ | | | | | | ✓ | |
| Matek and Zebec [72] | | ✓ | ✓ | | | | | | | | | | | ✓ | | |
| Yu et al. [113] | | | | ✓ | | | | ✓ | | ✓ | | | | | | |
| Koskela et al. [63] | ✓ | ✓ | ✓ | | | | | ✓ | | ✓ | | | | | | |
| Zhang et al. [117] | | ✓ | | | ✓ | ✓ | | | | | ✓ | | | | | |
| Suchal and Návrat [101] | ✓ | ✓ | ✓ | | | | | | | ✓ | ✓ | | | | | |
| Nossa proposta | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | ✓ |

No contexto de recomendações no GitHub, encontramos alguns estudos como o de Suchal and Návrat [101], que assumiram a tarefa de prever repositórios para os usuários do GitHub. Neste estudo, os autores propuseram um sistema de recomendação baseado nos $k$ vizinhos mais próximos, analisando as linguagens de programação e código-fonte para prever repositórios que os usuários do GitHub gostariam de observar. No nosso estudo não aplicamos nenhuma análise baseada nos textos dos códigos-fonte, pois a base de dados que

utilizamos para realização dos experimentos (GHTorrent[2]) não contém o texto do código-fonte, apenas informa qual a linguagem de programação (LP) usada no projeto. Entretanto, temos uma abrangência maior de características utilizadas no nosso trabalho, que Suchal and Návrat [101] não chega a analisar.

Na linha dos trabalhos de Orii [79] e Koskela et al. [63], identificamos que eles realizaram recomendações de projetos baseadas na análise dos repositórios alinhado à LP aplicada. O primeiro utiliza um algoritmo que combina fatoração de matriz probabilística e modelagem probabilística de tópicos; o segundo trabalho utiliza um método híbrido, combinando três diferentes medidas de similaridade com três diferentes conjuntos de características para gerar uma lista única de recomendações.

Nosso trabalho tem uma semelhança com os estudos de Orii [79] e Koskela et al. [63], no que diz respeito à exploração da linguagem de programação, que pode ser identificada por meio de tags ou recursos categóricos [33] [58] [90]. No nosso estudo, exploramos a informação sobre a principal LP utilizada por cada repositório e a principal LP de preferência do usuário, para recomendarmos repositórios com linguagens de programação semelhantes às LPs utilizadas pelos usuários. No trabalho de Orii [79] e Koskela et al. [63], eles utilizaram como uma de suas principais características apenas a informação principal sobre a linguagem de programação utilizada por cada repositório.

Outra característica utilizada em alguns estudo é a informações do quantitativo de *pull-request* aplicados aos repositórios ou a informação do revisor que realizou o *pull-request*. O *pull-request* permite que o repositório receba contribuições para melhoria de suas funcionalidades. Por este motivo, nosso estudo inclui o *pull-request*, para que possamos conhecer das alterações realizadas nos repositórios da plataforma. No estudo aplicado por Yu et al. [113] foram combinadas informações de usuários, projetos e a análise de revisão de *pull-requests* para recomendar novos revisores. Este trabalho cria uma rede de análise de revisão de *pull-request*, para que as revisões sejam realizadas a contento para a possível concordância da alteração por parte do revisor do *pull-request*. Os autores criaram uma rede de comentários dos desenvolvedores que responderam os *issues* dos projetos em evidência. Em um trabalho mais recente e dando continuidade ao trabalho anterior, Yu et al. [115] acrescentaram à rede de comentários informações sobre o expertise de cada revisor e os interesses

[2]www.ghtorrent.org

comuns entre eles, melhorando significativamente a precisão das recomendações de revisores de *pull-resquests*. No nosso trabalho, não chegamos a detalhar o *pull-request* até uma análise de revisão. Apenas verificamos a quantidade de *pull-request* aceitos em um projeto para conhecermos das atualizações dos repositórios.

Alguns estudos utilizam métodos que trabalham com estrutura de rede, como tratam os trabalhos de Lee and Chen [65] e Matek and Zebec [72]. Os sistemas de recomendação de análise de rede são geralmente baseados em técnicas de previsão de links. Esses estudos observaram os eventos que os usuários contribuíram em um intervalo específico de tempo, como exemplo *PushEvent* e *PullRequestEvent*, e eventos que os usuários interagiram na plataforma como *PushEvent*, *PullRequestEvent*, *ForkEvent*, *IssueEvent*, *IssueCommentEvent* e *WatchEvent*, para realizar recomendações de repositórios. Com tais informações, eles analisaram a semelhança dos repositórios baseada na quantidade de eventos. No nosso estudo, utilizamos as informações de eventos não apenas como um quantitativo por repositório, mas também foi essencial para compreendermos melhor as atividades dos projetos e percebermos se estes recebiam muitas contribuições ou não, seja durante a vida do projeto ou diariamente, por exemplo.

Recentemente o estudo de Liu et al. [68] propôs mitigar os esforços dos desenvolvedores em encontrar repositórios utilizando o modelo NNLRank (*Neural Network for List-wise Ranking*). O NNLRank aproveita as características dos projetos e a experiência dos desenvolvedores para recomendar repositórios de acordo com as preferências dos usuários. O NNLRank utiliza uma rede neural como uma função de ranqueamento, que permite processar uma lista de projetos candidatos, utilizando as características de LP, projetos que o usuário observa e as organizações que ele participa. Outro estudo investigou a viabilidade de alavancar a preferência da linguagem de programação do usuário para melhorar o desempenho das recomendações de repositórios baseada em OCCF (*One-Class Collaborative Filtering*) [58]. Com base na fatoração de matriz, propuseram a fatoração de matriz regularizada pela linguagem (LRMF), que trata as relações entre as preferências e conhecimento nas linguagens de programação do usuário. No nosso estudo, não utilizamos redes neurais para recomendação de repositórios, apenas utilizamos como uma das características, a linguagem de programação, que é aplicada nos dois trabalhos citados acima.

Finalmente, tem-se falado bastante no uso de sistemas de recomendação para diferentes

aplicações e o uso de distintas técnicas ou combinação delas para a acurácia dos recomendadores. Diversas análises sobre o ambiente GitHub e suas características têm sido realizadas para entender o comportamento dos usuários dentro de um ambiente colaborativo e como estimular cada vez mais as contribuições. A pesquisa reportada neste documento gastou esforços no sentido de tentar compreender o comportamento dos usuários do GitHub, onde utilizamos a maioria das características utilizadas em outros estudos, a saber: (linguagem de programação, *watchers*, *followers*, *pull-requests*, *fork*, *issues*, *commits*, *events*); e criamos outras características derivadas a partir de eventos gerados por ações dos usuários na plataforma, que chamamos de características sobre engajamento dos usuários nos projetos. O escopo deste trabalho concentrou-se nos modelos do estado-da-arte para aprendizagem de máquina e selecionamos àqueles que são amplamente utilizados na literatura para FBC e FC (Item-KNN, Máquina de Fatoração, Random Forest e XGBoost).

*Diante dos trabalhos analisados, verificamos que nenhum deles avaliou as diversas características de usuários e projetos selecionadas em nosso estudo, ou criaram atributos derivados dessas características, e que foram extraídas dos dados e metadados disponíveis pela plataforma GitHub. Aproveitando essas características e explorando as técnicas de aprendizado de máquina relacionadas especificamente à tarefa de classificação, pontuamos uma lista de projetos candidatos baseados na característica fork (que não foi utilizado em nenhum outro trabalho com este propósito). Muitos dos trabalhos apresentados também utilizaram técnicas de aprendizado. No nosso trabalho escolhemos utilizar Item-KNN, Máquina de Fatoração (que não foi utilizado em nenhum outro trabalho com este propósito), Random Forest e XGBoost. Desta forma, queremos avançar no conhecimento das características disponibilizadas pela plataforma GitHub, reconhecendo quais delas são mais relevantes para uma recomendação precisa e confirmar a eficácia dos modelos de aprendizado para classificação. Tudo isso recorrendo ao arcabouço teórico apresentado pela academia.*

## 3.3 Considerações Finais

Neste capítulo, apresentamos o ecossistema GitHub por meio de características sociais, colaboração e eventos. A organização desse ecossistema e as análises dos artigos apresentados neste capítulo podem desempenhar diversos papéis em pesquisas científicas e desenvolvi-

mento de sistemas recomendação ([92]), como, por exemplo: (i) servir como um guia de aspectos que devem ser considerados por pesquisadores e desenvolvedores, (ii) prover métricas de recomendação por meio dos quais os pesquisadores podem realizar novas descobertas e raciocinar sobre a área em estudo, e (iii) justificar e motivar o desenvolvimento de novas estratégias de modelos de recomendação. Todos os estudos analisados se concentram na recomendação de projetos direcionada para certas características, seja do usuário ou projeto. Na nossa proposta abordamos recomendações centradas numa diversidade de características como *atratividade*, *engajamento*, *medidas estatísticas* e *visibilidade*.

# Capítulo 4

# Exploração de Dados do GitHub

Um dos requisitos fundamentais para um SR na plataforma GitHub são os elementos bem definidos para usuários e itens. Tentando entender como seus usuários utilizam a plataforma na colaboração de software, realizamos um estudo empírico para entender as características dos repositórios e como os usuários tiram proveito dos principais recursos do GitHub (por exemplo, *commits*, *pull-requests*, linguagem de programação e outros). Nas seções a seguir, abordaremos as ferramentas utilizadas para coleta dos dados do GitHub (Seção 4.1) e exibiremos nossa análise dos dados do GitHub (Seção 4.2).

## 4.1 Coleta dos Dados

Os dados coletados neste trabalho foram disponibilizados por meio do projeto chamado GHTorrent [43], podendo ser acessados pelos serviços do GooGle BigQuery[1]. A API conhecida por Google BigQuery tem a importação dos dados atualizada no projeto GHTorrent MySQL, contendo todas as tabelas e dados relacionados ao projeto GHTorrent. Para complementar os dados analisados neste capítulo, usamos em combinação ao GHTorrent[2], o GitHub Archive[3]. Trabalhamos com a base de dados atualizada até a data de 01/04/2018 (versão mais atual disponível no BigQuery até o momento de escrita desta tese). Neste modelo de dados, o GitHub tem cerca de 24,2 milhões de usuários e 83,6 milhões de repositórios.

[1] https://cloud.google.com/bigquery/

[2] www.ghtorrent.org

[3] www.githubarchive.org

## 4.2 Análise dos Dados

O GitHub permite que os desenvolvedores hospedem seus projetos e códigos-fonte, realizem suas revisões colaborativas de código-fonte, realizem o rastreamento integrado de problemas, dentre outras funcionalidades. Tudo isso por meio de recursos sociais integrados disponibilizados pela plataforma. Os usuários também têm perfis que podem ser preenchidos com informações de identificação e contêm suas atividades recentes no site. Neste estudo, iremos apresentar alguns dados coletados do GitHub, de acordo com a base de dados utilizada.

Analisando a plataforma até abril de 2018, temos a totalidade de 24.2 milhões de usuários e 83.6 milhões de projetos cadastrados no GitHub. Desta totalidade, apenas 11.9 milhões de usuários e 48.4 dos repositórios estão ativos. O termo "evento" significa que ocorreu uma atividade relacionada ao usuário ou repositório (o GitHub contém mais de 30 tipos de eventos diferentes). Por exemplo, atualização de código (*commit*), repositório que passou a ser observado (*watcher*) ou problema que foi aberto, que é chamado de (*issue*). Dentre os usuários, temos o conceito de organizações, onde identificamos 417 mil organizações ativas. E, em relação aos repositórios, temos um número expressivo de problemas a serem tratados nos repositórios, chegando a ter 6.9 milhões de problemas aguardando uma solução pelos usuários contribuidores.

A plataforma GitHub define os repositórios que apresentaram atividade ou repositórios que apresentaram inatividade, a partir de uma alteração de status do repositório. Repositórios deletados da plataforma não são deletados fisicamente, eles passam a ter o status de inativo. Para que não utilizássemos repositórios deletados, ou seja, inativos na plataforma, selecionamos apenas o projetos com status ativo na plataforma. Entretanto, no nosso trabalho, não tratamos um projeto ativo apenas como um projeto que não foi deletado na plataforma GitHub. Neste trabalho, tratamos um repositório que apresentaram atividade como àquele em que a data do último evento realizado no repositório é diferente da data de criação do mesmo.

A Tabela 4.1 exibe as top-10 das organizações mais ativas, de acordo com o número de eventos que recebeu, e repositórios mais atualizados, onde listamos a quantidade de eventos, contribuições e principal linguagem de programação associada a cada repositório. Como podemos observar, não necessariamente temos uma relação direta entre quantidade de con-

tribuidores e eventos para um projeto. Por exemplo, temos o projeto *Microsoft/vscode*, que possui apenas 334 contribuidores e tem a maior quantidade de eventos (12.607). Em comparação com o projeto *jlord/patchwork*, que possui 12.944 contribuidores e não se encontra nem entre os três primeiros projetos em contribuição.

Tabela 4.1: Organizações (O) e Repositórios (P) com seus Eventos por Contribuidores

| **Organizações** | **Eventos (O)** | **Repositórios** | **Eventos (P)** | **Contribuidores** | **Linguagem** |
|---|---|---|---|---|---|
| Microsoft | 31928 | Microsoft/vscode | 12607 | 334 | TypeScript |
| facebook | 19575 | facebook/react-native | 8371 | 1367 | JavaScript |
| angular | 16520 | FortAwesome/Font-Awesome | 7984 | 82 | HTML |
| docker | 15033 | docker/docker | 7974 | 1695 | Go |
| google | 13513 | angular/angular-cli | 6492 | 262 | TypeScript |
| learn-co-students | 10602 | npm/npm | 6435 | 432 | JavaScript |
| apache | 7927 | tensorflow/tensorflow | 6393 | 935 | C++ |
| driftyco | 7667 | jlord/patchwork | 6360 | 12944 | HTML |
| FortAwesome | 7215 | angular/angular | 5779 | 462 | TypeScript |
| elastic | 6975 | ansible/ansible | 5409 | 2770 | Python |

De acordo com a pesquisa realizada no estudo de Borges [15], a popularidade de um projeto de software é uma informação valiosa para os desenvolvedores de software de código aberto. Esses desenvolvedores desejam constantemente saber se seus repositórios estão atraindo novos seguidores, se novos projetos estão tendo aceitação ou se estão atendendo às expectativas de outros usuários para colaboração. O estudo realizou uma pesquisa perguntando para os usuários do GitHub qual a utilidade das seguintes métricas (*watchers*, *fork* e *stars*) para avaliar a popularidade dos projetos do GitHub. A característica *fork* ficou em segundo lugar, com 72% de respostas. Demonstrando ser uma característica relevante para a popularidade dos projetos, *stars* ficou em primeiro lugar com 83% e *watchers* em terceiro lugar com 67%. No nosso estudo, não utilizamos a característica *stars* (característica de popularidade do repositório), pois esta não se encontra disponível no conjunto de dados disponível pelo projeto GHTorrent.

Nossa intenção foi analisar as principais características de usuários e projetos para identificarmos quais delas seriam mais representativas para recomendação de projetos na plataforma GitHub. Como dito, o GitHub possui cerca de 30 eventos diferentes. Neste estudo focamos nos eventos mais representativos da plataforma: *PushEvent*, *IssueEvent*, *ForkEvent*,

*PullRequestEvent* e *WatchEvent*, pois são os eventos mais recorrentes no GitHub. As características relacionados aos eventos citados estão descritas na Tabela 2.3 do Capítulo 2. A característica *fork* foi utilizada para representar não apenas a popularidade de um projeto, mas também demonstrar se um projeto é alvo de contribuição por parte dos usuários ou não.

### Linguagem de Programação

As linguagens podem tratar de marcação, estilo, consulta estruturada, dentre outras [59]. Neste trabalho, tratamos Linguagem de Programação - LP como esse conjunto de linguagens. A LP é uma das restrições para codificação social e o seu uso por um colaborador depende das habilidades e experiências de codificação do profissional. Atualmente, a plataforma GitHub contém 353 linguagens de programação diferentes. A LP é parte essencial dos repositórios, pois o desenvolvedor, invariavelmente, precisa definir a LP que será utilizada na implementação do projeto, como também, a linguagem de programação identifica as relações entre os projetos [74]. Por exemplo, linguagens interpretadas, como Python, são adotadas devido à sua conveniência; outras, como C, são utilizadas devido à eficiência de execução. Portanto, a definição da LP pelo usuário implica que teremos uma limitação de repositórios.

A plataforma GitHub permite a criação de qualquer tipo de repositório. Muitos usuários utilizam a plataforma para outros fins, que não seja apenas para o desenvolvimento de software. Para evitar a análise de projetos que não sejam projetos de software (por exemplo, projetos de documentação simples) [59], filtramos os repositórios pelas linguagens de programação. Dentre os 83.6 milhões de projetos, temos 77 milhões de projetos sinalizados como projetos de software e que são ativos (projetos não excluídos) na plataforma, como exibido pela Figura 4.1. JavaScript é a linguagem mais popular (13%), seguida por Java (7,7%), Python (5,8%), Ruby (4,8%) e HTML (3,8%). Dada a quantidade massiva de dados na plataforma, limitamos o nosso conjunto de dados para experimentação aos projetos de software que contém as 20 (vinte) linguagens de programação mais utilizadas, pois estas alcançam cerca de 55,7% dos projetos de software e ativos da plataforma.

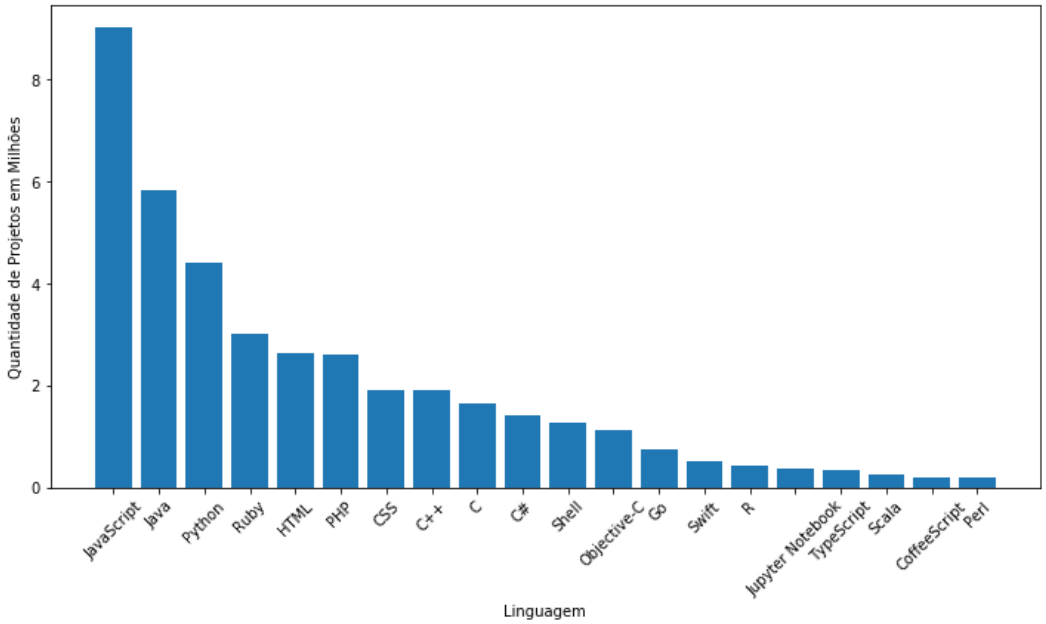


Figura 4.1: Linguagens de Programação Mais Utilizadas

### *Commits*

Quando analisamos os projetos que recebem mais *commits* e suas respectivas linguagens de programação, vemos que as três primeiras linguagens com o maior número de *commits* por projeto correspondem às três primeiras linguagens mais utilizadas no GitHub, como mostra a Figura 4.1. Entretanto, a demais linguagens mudam um pouco de posição, de acordo com a preferência do usuário na atualização dos projetos, como mostra a Figura 4.2. Ainda, analisando as linguagens de programação para os projetos que mais recebem *fork*, também identificamos que JavaScript, Java e Python estão entre as três primeiras linguagens, demonstrando que os usuários têm uma preferência significativa por estas linguagens.

A atividade dos projetos no Github é refletida, principalmente, pela quantidade de *commits* que um projeto recebe por um período. Assim, podemos medir a atividade de um projeto usando duas diferentes abordagens: período pelo qual os *commits* são realizados e pela quantidade de *commits* que o projeto recebe. Assim, para medir a atividade de um projeto, recuperamos a data de criação do projeto e os *commits* aplicados durante um período de tempo. Na Figura 4.3 podemos visualizar a quantidade de *commits* contados a partir do primeiro dia de criação do projeto até os trinta primeiros dias do projeto. Verificamos que a maioria dos projetos recebem muitos *commits* no período inicial do projeto e, após alguns

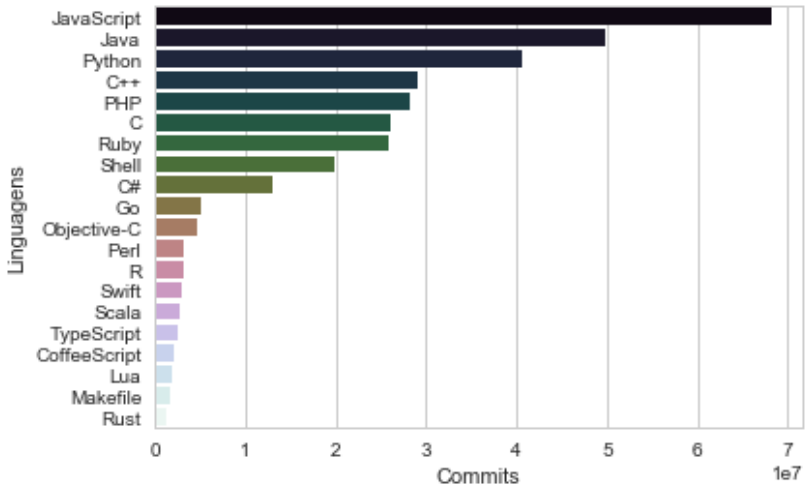


Figura 4.2: Linguagens de Programação com mais *commits*

meses, a quantidade de *commits* diminui consideravelmente. Por exemplo, depois de 06 (seis) meses, apenas 61% dos projetos continuavam sendo atualizados. No geral, a média no número de *commits* por projeto é de 6 (seis) e 90% dos projetos têm menos que 50 *commits*. Existe um grande número de projetos com pouca atividade e os projetos mais ativos são aqueles responsáveis pela maioria dos *commits* na plataforma.

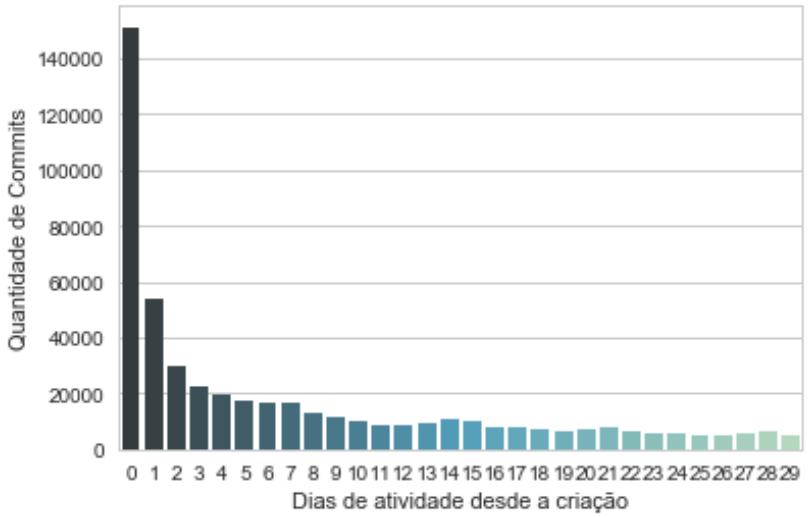


Figura 4.3: *Commits* Diários dos Projetos

Como podemos verificar, muitos projetos recebem poucos *commits* e tornam-se obsoletos depois de um período de tempo. Analisando a data de criação dos projetos e a última data de atualização destes, em média, os projetos são atualizados mais intensamente nos seus 10 (dez) primeiros dias. Sendo assim, podemos observar que muitos projetos são adicionados na

plataforma apenas para armazenamento dos arquivos dos repositórios ou para serem testados por um período curto de tempo.

### ***Forks* e *Pull-Requests***

No modelo de desenvolvimento de projetos de software de forma colaborativa, os repositórios dividem-se em: repositórios base (principal) e repositórios *fork* (clone do repositório base). Em média, temos 19,2% dos projetos da plataforma que são forks dos repositórios base. As dez principais linguagens de programação dos projetos que são *fork* continuam semelhantes às principais linguagens da plataforma para os projetos com maior número de *commits* e *pull-requests*, que são: JavaScript, Java, Python, HTML, PHP, CSS, C++ e C. Para as demais linguagens, ocorre uma variação no uso das mesmas, mas sem muita representatividade. Na Tabela 4.2, exibimos os top-10 repositórios que apresentaram a maior quantidade de *forks* e que contêm as dez linguagens de programação mais utilizadas na plataforma. Como podemos observar, os principais repositórios que têm mais *forks* são guias para compartilhamento de dados para aplicação de ciência de dados ou bibliotecas e frameworks de desenvolvimento de software.

Tabela 4.2: Repositórios com maior número de *forks* até abril de 2018

| **Repositórios** | **Quantidade de *Forks*** |
|---|---|
| jtleek/datasharing | 23320 |
| tensorflow/tensorflow | 21080 |
| octocat/Spoon-Knife | 18841 |
| rdpeng/ProgrammingAssignment | 16069 |
| twbs/bootstrap | 14809 |
| SmartThingsCommunity/SmartThingsPublic | 13792 |
| barryclark/jekyll-now | 10479 |
| udacity/frontend-nanodegree-resume | 10407 |
| github/gitignore | 10121 |
| LarryMad/recipes | 10074 |

A atividade de um repositório *fork* é registrada independentemente da sua associação com o repositório base. Quando um *commit* é feito no repositório *fork* e atualizado via solicitação de *pull-request*, este *commit* não aparece no repositório base. Portanto, muitas atividades dos repositórios *forks* (por exemplo, *commits*, *pull-requests*) não aparecem na contagem final de *commits* do repositório base e, de acordo com o modelo de desenvolvimento *"Fork & Pull"* do GitHub [59], para que existam *pull-requests* é necessário que o usuário tenha realizado, inicialmente, um *fork* do repositório base. Por padrão, as solicitações de *pull-requests* são enviadas ao repositório base para revisão de código, que envolve dois tipos de comentários:

- Gerais (Discussão): comentários gerais sobre o conteúdo do *pull-request*. As partes interessadas discutem sobre informações técnicas para adequação do *pull-request* no repositório base.

- In-Line (Revisão de código): comentários sobre seções específicas do código. O revisor faz anotações de confirmação, em sua maioria de natureza técnica, para identificar possíveis melhorias no repositório base.

Esse modelo aprimora o processo de revisão, integrando análises de código, discussões e problemas. Solicitações combinadas, de *fork* e *pull-request*, criam um novo modelo de desenvolvimento, no qual as alterações são enviadas aos proprietários dos projetos e passam por uma revisão de código antes da integração do repositório base, melhorando a qualidade dos projetos. Na Figura 4.4, podemos verificar que a quantidade de *pull-requests* aplicado nos repositórios por ano tem aumentado consideravelmente, demonstrando que os repositórios estão recebendo cada vez mais atualizações. Como podemos ver, no ano de 2018, foram mais de 30 milhões de *pull-requests*. Para que isso aconteça, é necessário que *forks* tenham sido realizados previamente. Na Figura 4.4, no ano de 2018 o valor encontra-se abaixo dos demais devido aos dados serem coletados apenas até o mês de abril de 2018.

Na Figura 4.5, comparamos a quantidade de *commits* e *pull-requests* aplicados aos repositórios base e aos repositórios *fork*. Selecionamos mil projetos com o maior número de *commits* e que tenham *forks* relacionados a eles para que pudéssemos analisar o comportamento de *forks* e *pull-requests*. Podemos verificar que os repositórios base possuem uma maior quantidade de *pull-requests* e *commits*. Esse comportamento, no geral, apresenta-se na normalidade, pois a quantidade de *pull-requests* nos indica que os projetos *forks* estão

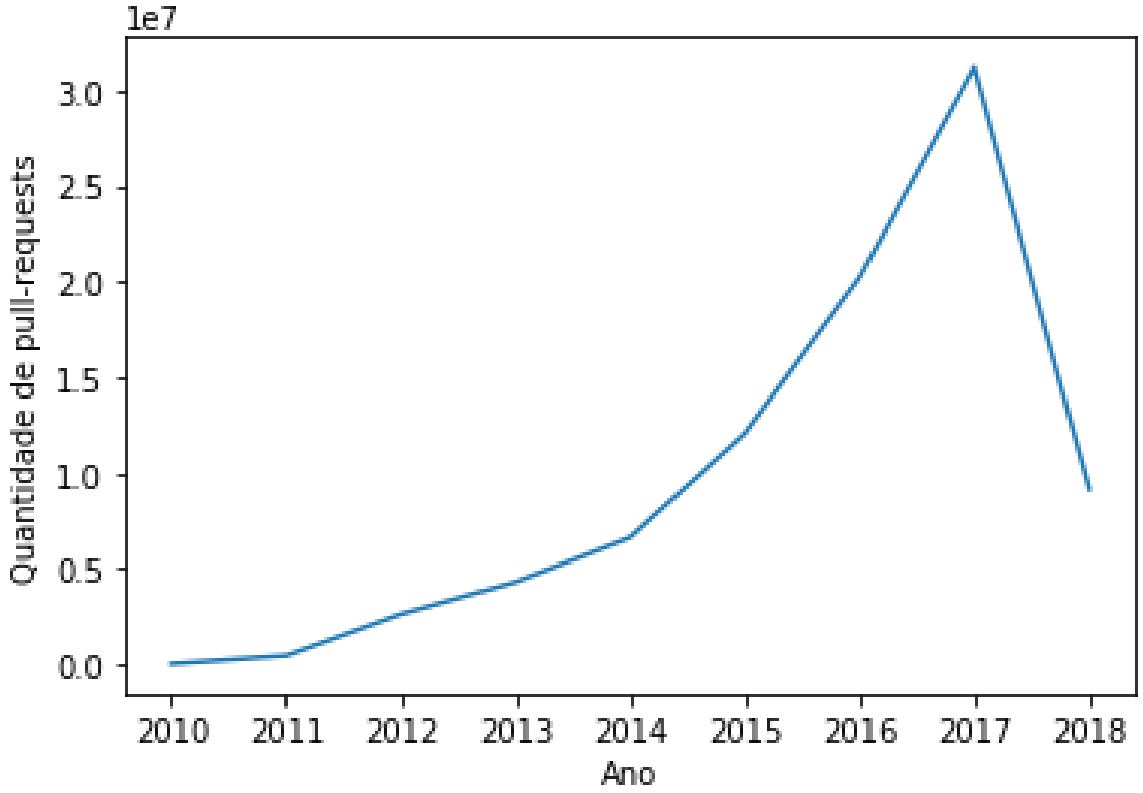


Figura 4.4: Crescimento da quantidade de *pull-requests* por ano

sendo utilizados para colaboração e melhorias no projeto base. Também podemos verificar que os repositórios *fork* tem, em alguns casos, mais *commits* que os próprios repositórios base. Onde podemos sugerir que muitos dos projetos base são adicionados na plataforma após terem sido desenvolvidos ou também poderia ser pelo fato de que os *commits* não são contabilizados para os projetos base. Além disso, podemos concluir a relevância do uso dos projetos *fork* como classificador em nossa proposta de trabalho. Por isso, precisamos selecionar os projetos que apresentam ser os mais relevantes para os usuários, para que possamos estimular a realização de *forks* e aumentar a colaboração dos projetos na plataforma. Um dos repositórios que possue o maior número de *forks* é o *octocat/Spoon-Knife*, que é um repositório administrado pelo próprio GitHub para os usuários testarem como funciona o *fork*.

De acordo com a análise realizada, podemos verificar que o uso da plataforma GitHub tem crescido em um ritmo acelerado e, cada vez mais, seus usuários têm buscado maneiras inovadoras de explorá-la. Desta forma, o modelo de desenvolvimento de software aplicado no GitHub torna-se cada vez mais utilizado, mostrando-se uma fonte atraente para a pesquisa

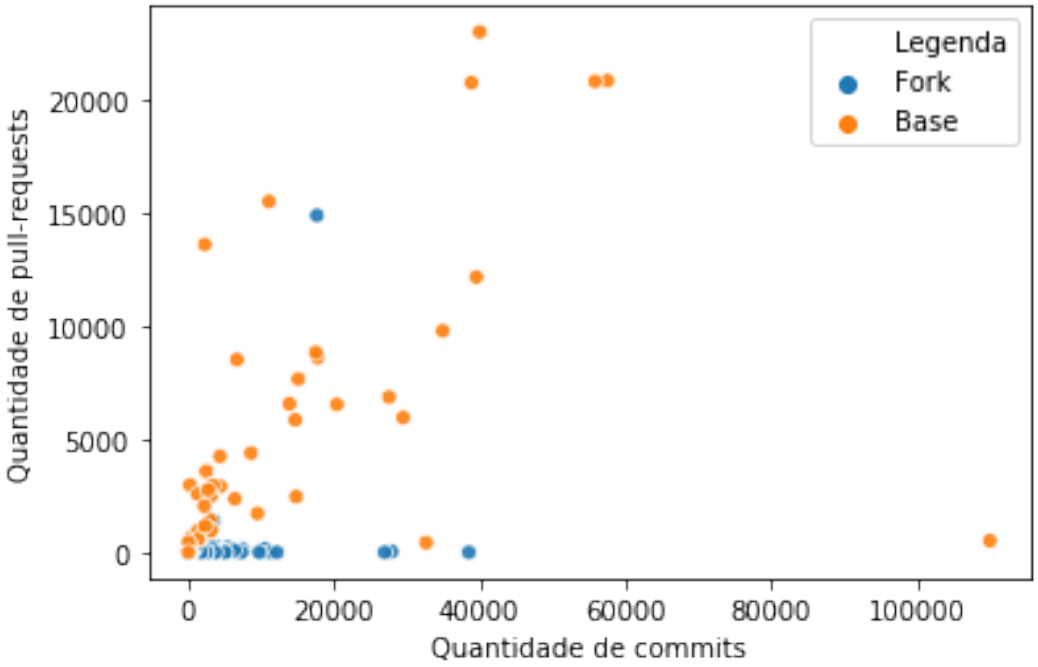


Figura 4.5: Comparação dos repositórios base e repositórios *fork* em relação à quantidade de *commits* e *pull-requests*

em engenharia de software. Por isso, decidimos por utilizar as principais características pessoais e sociais da plataforma, como também propomos no nosso trabalho novas métricas de atividade do projeto para conhecermos um pouco mais sobre o compartamento dos usuários e seus respectivos projetos na plataforma GitHub.

## 4.3 Considerações Finais

Neste capítulo, exploramos os dados do ecossistema GitHub por meio de características individuais, sociais e de colaboração dos usuários e suas respectivas atividades de projetos. Por meio da exploração dos dados, identificamos a necessidade de recomendação de projetos baseados na característica alvo chamada *fork*. A partir dessa informação, conseguimos identificar a possibilidade de crescimento na colaboração de projetos entre usuários. Demonstramos a importância dos repositórios, seu crescimento e atualizações dentro da plataforma GitHub. O estudo aplicado neste capítulo demonstrou a viabilidade do uso da característica *fork* como alvo no aprendizado de máquina, como também, demonstrou a viabilidade do uso das demais características da plataforma no estudo de engenharia de software.

# Capítulo 5

# Metodologia Aplicada

O processo de mineração de dados geralmente consiste em 3 etapas: Pré-processamento de Dados, Análise de Dados e Interpretação de Resultados. Na primeira etapa, parcialmente descrita no Capítulo 4, realizamos a coleta e modelagem dos dados dos usuários e projetos no ambiente do GitHub. Durante a análise de comportamento dos usuários e projetos, não apenas utilizamos as características semelhantes entre usuários e projetos, verificamos a necessidade de uma melhor compreensão do comportamento destas características e como elas evoluem no tempo. Neste capítulo, apresentamos outras contribuições nesta direção, analisando um amplo conjunto de características da plataforma e propomos outras derivadas a partir das características selecionadas, a fim de investigar alguns fatores que se correlacionam com as preferências dos usuários por projetos.

Durante a segunda etapa, realizamos a análise dos dados por meio da aplicação do estado da arte dos modelos de aprendizado de máquina, disponíveis na literatura, ao conjunto de dados selecionado. Por fim, na terceira etapa, realizamos a interpretação dos resultados, onde apresentamos a análise dos principais resultados dos experimentos e discutimos nossas descobertas sob a perspectiva de nossas perguntas de pesquisa. Estas duas etapas serão discutidas no Capítulo 6.

Neste capítulo descrevemos as estratégias de pré-processamento dos dados e a metodologia de avaliação aplicada, incluindo os conjuntos de dados, modelos de classificação utilizados e métricas aplicadas. Nas seções a seguir, primeiro apresentamos uma visão geral do processo de trabalho e aprendizagem dos modelos (Seção 5.1). Em seguida, apresentamos o pré-processamento dos dados propostos (Seção 5.2). Depois, descrevemos as características

selecionadas para o conjunto de dados utilizado no trabalho (Seção 5.3). Logo em seguida, propomos atributos derivados a partir de algumas características coletadas (Seção 5.4). Após isso, contextualizamos as abordagens clássicas para sistemas de recomendação (Seção 2.2). Por fim, apresentamos as métricas de avaliação a serem aplicadas (Seção 5.6).

## 5.1 Visão Geral

Para o processo de mineração de dados, a Figura 5.1 apresenta os 06 (seis) passos correspondentes ao processo de mineração de dados. O usuário é o ponto central do processo, dando início a partir do passo 01 (um) e finalizando o processo no passo 06 (seis). No nosso estudo, aplicamos a recomendação de projeto offline, ou seja, seguimos os passos de 01 (um) à 05 (cinco), não aplicando o passo 06 (seis), onde é recebido o feeedback do usuário em relação as recomendações propostas pelo recomendador. Entretanto, apesar de não apresentarmos uma recomendação on-line, geramos a recomendação neste trabalho. Utilizamos dois grandes conjuntos de dados gerados a partir de dados temporais offline. Recuperamos dados de usuários e projetos no período de tempo de 05 (cinco) anos (cerca de 10 mil usuários e 500 mil projetos), pois seria inviável, por questões de processamento, analisar todos os projetos da plataforma desde sua criação (atualmente a plataforma encontra-se com cerca de 83 milhões de projetos).

Detalhando a visão geral exibida na Figura 5.1, iniciamos, a partir do passo 01 (um), analisando o ambiente GitHub para conhecer sobre os recursos disponíveis pela plataforma e forma de colaboração social entre os usuários. Partimos do rastreamento das características dos usuários e projetos, interações entre usuários, semelhança com outros projetos, comportamentos passados, dentre outros, para podermos oferecer recomendações personalizadas e criar experiências incomparáveis para os usuários. A etapa do pré-processamento definida no passo 02 (dois), onde a partir da análise do ambiente GitHub e identificadas as características mais relevantes para o nosso estudo, coletamos os dados dos usuários e seus respectivos projetos e criamos grupos que contextualizam essas características, definindo o perfil do usuário e o perfil do projeto.

Passando para a etapa de análise de dados, a tecnologia de autoaprendizagem adapta-se às necessidades e expectativas sempre em mudança dos usuários e seus respectivos projetos,

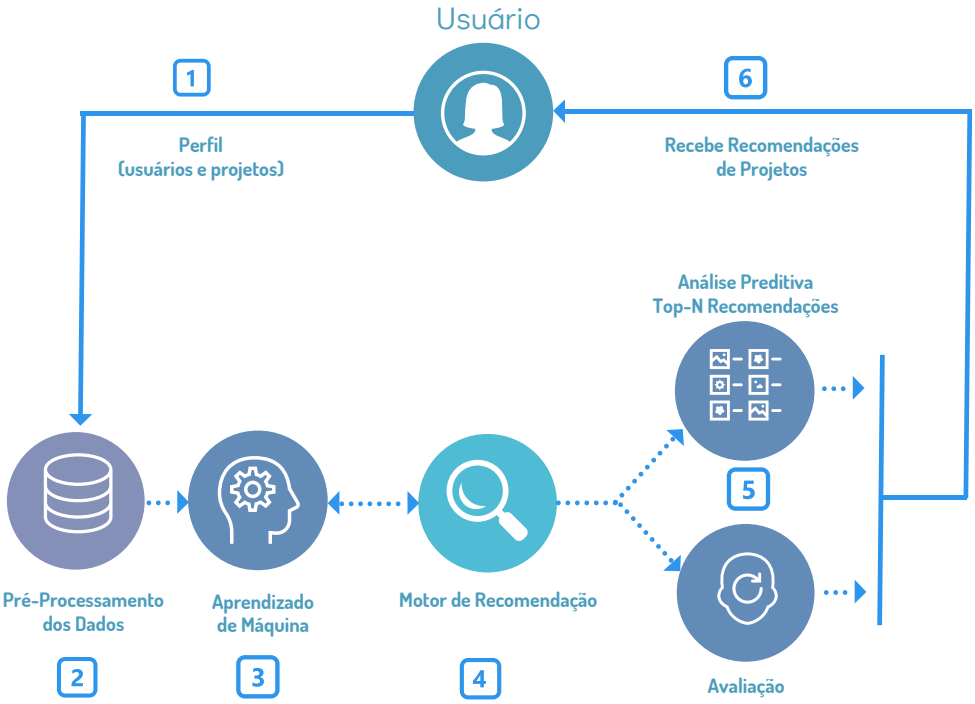


Figura 5.1: Visão Geral

resultando em maior relevância nas recomendações de projetos. Portanto, aplicamos os algoritmos de aprendizagem ao mapa de interesses dos usuários e projetos, como sinalizado no passo 03 (três). Neste trabalho foram exploradas técnicas de aprendizado de máquina relacionadas especificamente à tarefa de classificação, a qual se dá por meio de um algoritmo, cujo objetivo é predizer o rótulo (classe) de novas entradas com base nos exemplos de entrada já rotulados. De modo geral, o objetivo do trabalho é experimentar um conjunto de algoritmos de diferentes paradigmas aplicados aos perfis de usuários e projetos, que são combinados para realizar a predição. Após isso, é verificada a precisão da predição que determina o sucesso do recomendador.

Como constam nos passos 04 (quatro) e 05 (cinco), a partir da aprendizagem de máquina, aplicamos a predição, que calcula a pontuação de relevância do projeto para o usuário com base em todo o contexto determinado. Essa pontuação define um *score* de classificação do projeto como candidato à *fork* ou não. Avaliamos a qualidade das predições fornecidas pelos modelos e comparamos entre si a partir dos fatores de qualidade da recomendação utilizando as métricas: (a) *precision* e (b) *recall*. A partir da pontuação, podemos criar uma lista

descrescente, que nos informa a posição de cada projeto na recomendação. Por fim, como explicado no passo 06 (seis), ficamos habilitados a enviar o top-N projetos recomendados para o usuário, onde N representa o número de projetos a serem recomendados. Nas seções a seguir, detalhamos a metodologia aplicada no nosso processo de apredizagem de máquina e recomendação.

## 5.2 Pré-Processamento de Dados

Embora a plataforma GitHub tenha muitos usuários que participem da codificação social, é sabido que ela possui usuários e projetos inativos, ou usuários que não participam do desenvolvimento de software em equipe. Além disso, a maioria dos usuários do GitHub não possui um registro significativo de *commits* para que possamos tirar conclusões relevantes sobre o comportamento de contribuições de projetos [21]. Portanto, nosso estudo se concentrou em desenvolvedores de software que contribuíram em vários projetos, seja por meio de *commits* ou *pull-requests*. Usamos os dados do projeto GHTorrent[1] datado de 01/04/2018. Recuperamos dados no período de tempo de 05 (cinco) anos, onde selecionamos 10 mil desenvolvedores de software ativos com histórico de contribuições no GitHub. Para cada desenvolvedor, recuperamos cerca de 50 projetos em que eles realizaram mais contribuições e que contém também projetos que são *fork* de outros projetos. Temos 9.580 projetos semelhantes entre os usuários. O conjunto de dados utilizado neste estudo encontra-se na plataforma Zenodo[2].

Realizamos o experimento de forma offline, onde utilizamos um conjunto de dados pré-coletados. Não realizamos nenhuma interação on-line com os usuários. Os sistemas de recomendação são classificados de acordo com o método de predição utilizado. A predição de avaliações dos projetos ainda não avaliados pode ser feita de diferentes formas, utilizando métodos de aprendizado de máquinas, teorias de aproximação ou tipos de heurísticas. Neste trabalho, aplicamos os modelos de classificação selecionados ao conjunto de dados pré-coletado para aprendizado de máquina e recomendação dos projetos.

[1] www.ghtorrent.org

[2] https://zenodo.org/record/3840472

### 5.2.1 Amostragem

A amostragem é uma técnica da estatística utilizada para selecionar um subconjunto de dados relevante a partir de um grande conjunto de dados. Geralmente, a amostragem é utilizada por se mostrar como uma abordagem de baixo custo, pois o processamento de todo o conjunto de dados é computacionalmente muito caro [6]. A pesquisa partiu da investigação de qual seria o conjunto de dados mais adequado e que contenha as informações previamente definidas para à avaliação bem-sucedida do algoritmo de recomendação. Por isso, escolhemos dois tipos de amostragens, que serão apresentadas abaixo, contendo uma distribuição semelhante ao conjunto de dados original da plataforma GitHub.

A partir dessas amostragens, podemos criar conjuntos de dados de treinamento ($D^{Train}$) e teste ($D^{Test}$), particionados adequadamente a partir do conjunto de dados original e que podem ser utilizados nas fases de pré-processamento e validação dos dados. Geralmente, o conjunto de dados de treinamento é usado para aprender os parâmetros ou configurar os algoritmos usados na etapa de treinamento, enquanto o conjunto de dados de teste é usado para avaliar o modelo ou a configuração obtida na fase de treinamento, garantindo um bom desempenho com dados não treinados anteriormente. Ou seja, dados de treinamento não são utilizados em teste. [28]. Recuperamos dados de usuários e projetos no período de tempo de 05 (cinco) anos, pois seria inviável, por questões de processamento, analisar todos os projetos da plataforma desde sua criação (atualmente a plataforma encontra-se com cerca de 83 milhões de projetos).

Essa técnica de validação, onde dividimos o conjunto de dados original em treino e teste, pode ser feita seguindo proporções como 60/40, 70/30, 80/20 e outros. Decidimos por utilizar as amostragens na proporção 80/20 ao separar os conjuntos de dados de treinamento e teste. De acordo com [61] e [6], o particionamento no tamanho acima de 2/3 para o conjunto de treinamento é apropriado e é uma prática comum usar amostragem aleatória padrão sem substituição com uma proporção 80/20 ao separar os conjuntos de dados de treinamento e teste. Isso significa que, a partir da amostragem, selecionamos 20% das instâncias para o conjunto de testes e deixamos os 80% restantes para treinamento. Como dito anteriormente, selecionamos dois tipos de amostragens que chamamos de: i) amostragem balanceada e ii) amostragem negativa. No primeiro, buscamos selecionar o conjunto de dados de forma não aleatória, onde classes positivas são aproximadamente iguais aos valores negativos. Alguns

trabalhos indicam que distribuição natural geralmente não é a melhor distribuição para um classificador aprender. Entretanto, para alguns trabalhos, os estudos indicam que uma amostragem balanceada pode não ser a melhor opção [24] [80]. Por isso, o segundo conjunto de dados trata de uma amostragem desbalanceada, onde temos maiores observações negativas do que positivas, sendo mais próximo do que acontece no ambiente GitHub, como detalhado na explicação a seguir sobre amostragem negativa.

As características numéricas apresentadas em cada amostra foram normalizadas. A normalização significa dimensionar uma variável para ter valores entre 0 e 1. Para isso, utilizamos a Equação 5.1 para o redimensionamento dos valores numéricos das características. Para cada característica numérica, coletamos $X_{min}$, que representa o menor valor aplicado para a característica, e $X_{max}$, que representa o maior valor aplicado para a característica. Para cada valor numérico $X$ calculamos um novo valor numérico para a característica, que é representado por $X_{new}$.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \in (0, 1] \tag{5.1}$$

O objetivo da normalização foi fazer com que cada ponto de dados tivesse a mesma escala para que cada característica fosse igualmente importante [92]. Detalhamos a metodologia aplicada para as duas amostragens a seguir.

### Amostragem Balanceada

Em um conjunto de dados balanceado temos os valores positivos, que são aproximadamente iguais aos valores negativos, estando o conjunto de dados em equilíbrio. Balanceamos nossa amostra de acordo com o alvo, ou seja, projetos que são *forks* de outros projetos, onde consideramos os mesmos como positivos (*target* igual a 1), e projetos que não são *fork* de outros projetos foram considerados projetos negativos (*target* igual a 0).

Por exemplo, selecionamos aleatoriamente 50 projetos do usuário $u_1$. Utilizando a amostra na proporção 80/20, então, 40 projetos foram escolhidos para o conjunto de treino e os 10 restantes ficarão no conjunto de teste. No conjunto de treino, seguimos a regra do balanceamento nos 40 projetos selecionados, onde 20 deles são projetos que são *fork* de outros projetos (*target* igual a 1) e os outros 20 não são *fork* de outros projetos (*target* igual a 0), no caso, são projetos do próprio usuário. Para o conjunto de teste, mantivemos os projetos

de acordo com a realidade dos projetos do usuário (não estão balanceados) e que são mais recentes, contendo os projetos que são *fork* de outros projetos (*target* igual a 1) e que não são *fork* (*target* igual a 0).

Um outro ponto a ser observado é a linha do tempo dos projetos, pois, como recuperamos os projetos dos últimos 5 anos, a seleção dos 80/20 projetos estão de acordo com a linha do tempo dos projetos. Ou seja, os 10 projetos de teste são mais recentes que os 40 projetos de treino.

Por fim, se apenas escolhêssemos uma amostra de forma aleatória, sem balancear o conjunto de dados, correríamos o risco de ter poucos projetos que são *fork* de outros projetos. Como mencionado anteriormente, dentre os 83 milhões de repositórios, apenas 19,7% dos repositórios são *fork* de outros projetos até abril de 2018. Desta forma, o modelo não conseguiria aprender as preferências dos usuários por projetos para a realização de *forks*.

### Amostragem Negativa

Na amostragem negativa, a técnica é utilizada para treinar modelos de aprendizado de máquina que geralmente utiliza a técnica de maiores observações negativas do que positivas [111]. Na maioria dos casos, essas observações negativas não são dadas explicitamente, portanto, devem ser produzidas de alguma forma.

Assim, ao confiar no feedback implícito do usuário, podemos inferir que qualquer outro projeto que o usuário conheça e não tenha feito *fork* ou que desconheça dentro da plataforma é menos relevante que os projetos que ele tenha feito *fork*. Devido à grande quantidade de projetos contidos na plataforma GitHub, pode ser o caso dos projetos não terem sido encontrados ou não estarem disponíveis no momento da seleção feita pelo usuário. Neste caso, é necessário obter "projetos com *forks* negativos" (ou seja, projetos não marcados como *fork* pelo usuário). Então, para criação do conjunto de dados, seguimos o procedimento descrito abaixo para cada usuário:

1. Selecionamos um conjunto de dados positivos, que apenas contém projetos que tenham sido *fork* de outros projetos;

2. Selecionamos um conjunto de dados negativos, que apenas contém projetos que não tenham sido *fork* de outros projetos.

3. Nos conjuntos selecionados, para relacionar os projetos com algo que o usuário não tenha interesse, escolhemos os projetos que tenham a linguagem de programação principal diferente dos projetos escolhidos (usuário tenham feito *fork* em algum momento) pelo usuário.

4. Por fim, criamos um conjunto de dados final onde, para cada projeto do usuário contido no conjunto de dados positivos, adicionamos 09 (nove) projetos do conjunto de dados negativos.

O aumento do contraste entre os dados positivos e negativos desafiam a capacidade do modelo de distinguir entre projetos aparentemente semelhantes. Essa seleção de casos negativos tão distintos quanto possível dos positivos, faz com que os projetos sejam adequadamente agrupados no espaço vetorial. Por fim, observe que, adotando a técnica de amostragem negativa, geramos 02 (dois) diferentes tamanhos de amostras até chegarmos no conjunto de dados final. Escolhemos o tamanho de 09 (nove) projetos negativos e 01 (um) projeto positivo, pois é o mais próximo das condições reais dos sistemas de recomendação. Em geral, o usuário é exposto a uma lista de até 10 (dez) ofertas de filmes, livros e, no caso de nosso estudo, repositórios do GitHub, por exemplo, onde, entre as dez ofertas, ele escolhe uma delas.

Para o conjunto de teste, mantivemos os projetos de acordo com a realidade dos projetos do usuário (não estão balanceado negativamente) e que são mais recentes, contendo os projetos que são *fork* de outros projetos (*target* igual a 1) e que não são *fork* (*target* igual a 0).

## 5.3 Seleção das Características

Na plataforma GitHub, alguns estudos demonstram que os desenvolvedores de software preferem contribuir em projetos que eles tenham relacionamentos pré-existentes com os usuários proprietários e que tratem de assuntos de seu interesse na intenção de trocar experiências [47] [21]. Ainda na etapa de pré-processamento de dados, utilizando a base de dados indicada na Seção 4.1 descrita no Capítulo 4, para prevermos projetos candidatos para os desenvolvedores contribuírem, selecionamos um conjunto de características dos usuários e

dos projetos, incluindo o vínculo social do usuário com o proprietário de outros projetos, a linguagem de programação dos projetos, o tempo potencial em que o projeto recebeu contribuições por meio de *commits*, dentre outras. Ou seja, selecionamos um conjunto de características que contém uma variedade de tópicos e contextos dos usuários e dos projetos, que serão detalhados nesta seção.

Dabbish et al. [32] ressaltam que a visibilidade de algumas ações do GitHub, como seguir um usuário ou observar um projeto, levam a um aumento na qualidade das atualizações dos projetos pelos desenvolvedores (por exemplo, os desenvolvedores ficam motivados simplesmente porque eles sabem que alguém se importa com o seu trabalho). Além disso, essas ações encorajam o aumento das extensões desses projetos por outros desenvolvedores (*forks*), a exposição de outros projetos no portfólio de usuários, a geração de novas ideias, debates e o incentivo à codificação social.

Na plataforma GitHub, algumas dessas informações podem ser capturadas explicitamente a partir do feedback do usuário, o que inclui características explicitadas por ele em relação a outros projetos (por exemplo, seguir um usuário, observar outros projetos, ou atribuir estrela a um projeto no GitHub) na plataforma GitHub. Uma estratégia alternativa para capturar informações é confiar e capturar os dados sobre o comportamento do usuário sem explicitamente solicitá-los, ou seja, capturar o feedback implícito [51]. Por exemplo, um usuário que executou *fork* em muitos repositórios provavelmente tem conhecimento das linguagens de programação desses projetos ou pretende conhecer dessas linguagens de programação. Portanto, como muitos dos recursos disponibilizados na plataforma GitHub não têm feedback explícito, selecionamos as características dos usuários e projetos centradas tanto no feedback explícito (quantidade de observadores, seguidores, linguagem de programação principal e outras) como implícito (quantidade de *commits*, quantidade de *pull-requests*, quantidade de *issues* e outras).

Selecionamos algumas características da plataforma e propusemos outras derivadas a partir das características selecionadas, a fim de investigar alguns fatores que se correlacionam com as preferências dos usuários por projetos. As características selecionadas, como também as características derivadas são definidas de acordo com os conceitos de popularidade, contribuição, suporte e atividade dos projetos na linha do tempo. Alguns desses conceitos foram estudados em outros trabalhos relacionados ao comportamento dos usuários na plata-

forma GitHub [32] [21] [88] [87] [68]. Para melhor compreensão de cada tipo de variável e enquadramento das informações, separamos as características em grupos que chamamos de *atratividade*, *visibilidade*, *medidas estatísticas* e *engajamento*. Os dois primeiros grupos contêm características capturadas diretamente da plataforma (explícita ou implicitamente). Quanto ao terceiro e quarto grupos, as características foram derivadas a partir de eventos gerados por ações dos usuários na plataforma, criadas e propostas neste trabalho.

A seguir, apresentaremos cada grupo estabelecido e quais dados estão contidos a respeito das informações contextuais, como é o caso do grupo de atratividade, que são os fatores relacionados à popularidade dos usuários e projetos; de visibilidade, que é disponibilizada por uma plataforma que promove a contribuição e transparência da informação; do conjunto de engajamento, que foi definido a partir de métricas geradas a partir da atividade dos projetos na linha do tempo, sendo uma das contribuições deste trabalho; e do conjunto de estatísticas, que exibe a distribuição dos dados, que também é uma das contribuições deste trabalho. Esses conceitos se aplicam no contexto de usuários e projetos. Na Tabela 5.1, exibimos todas as características descritas nesta seção.

1. **Atratividade - ATR**: Os recursos de atratividade nos ajudam a analisar os fatores relacionados à popularidade e conexão entre os projetos. De acordo com Jarczyk et al. [57], quanto maior o número de observadores (*watchers*) em um projeto, maior o número de contribuições neste projeto. Além disso, é mais provável que os desenvolvedores contribuam em projetos que tenham a linguagem de programação semelhante aos projetos que eles contribuíram anteriormente. Assim, as linguagens de programação podem ser tratadas como informações similares para determinar os interesses dos usuários. Essas informações podem fazer a conexão para a relação entre usuários e projetos. Para efetividade de alguns modelos estudados, precisamos agrupar as linguagens de programação e, para isso, criamos a característica que chamamos de peso da linguagem (recurso derivado do cálculo da presença de uma linguagem de programação sobre todas as linguagens de programação usadas pelo usuário). Por isso, para cada projeto, capturamos a quantidade de observadores (*watchers*) e linguagens de programação; e, para cada usuário, capturamos a quantidade de seguidores (*followers*) e as preferências das linguagens de programação. Para cada usuário, calculamos o peso para cada linguagem de programação.

2. **Visibilidade - VIS**: A visibilidade trata da disponibilização de um ambiente que promove a transparência da informação. Os recursos de visibilidade dos repositórios, disponíveis na platafoma GitHub, foram reconhecidos como um aspecto valioso pela comunidade do GitHub [46]. A transparência permite que um usuário possa visualizar os projetos de seu interesse e ainda submeter alterações para este projeto, utilizando a funcionalidade de *fork*, ou fazer uma solicitação por meio de um *pull-request*. O desenvolvedor proprietário do repositório pode atualizar o repositório, por meio de *commits*, ou revisar as mudanças sugeridas no *pull-request* e decidir aceitar ou não tais alterações. Muitas discussões, conhecidas como *issues* podem ocorrer durante o processo de solicitação do *pull-request* ou quaisquer outros assuntos como esclarecimentos sobre o projetos, dicas de melhorias ou explicação da arquitetura. Para nosso estudo, utilizamos essas características para cada projeto e usuário. Coletamos o quantitativo de *pull-requests*, *commits* e *issues* por projeto. Também utilizamos a data de criação de cada projeto para transformarmos, a partir do campo *date*, as informações de dia, mês e ano.

3. **Medidas Estatísticas - ME**: Muito da compreensão sobre as características utilizadas está diretamente ligada aos detalhes sobre esses dados (por exemplo, a média e o desvio padrão). Como nossa capacidade de entendimento sobre o que está acontecendo em um ambiente é limitada, derivamos informações a partir de quatro características (*commits*, *pull-requests*, *watchers* e *issues*) para melhor compreendermos os dados e sua relevância no modelo de aprendizado. Criamos a distribuição dos dados para cada uma dessas características selecionadas e geramos os valores de média, mínimo, máximo e desvio padrão.

4. **Engajamento - ENG**: As atualizações dos projetos por seus proprietários ou por colaboradores incentivados à codificação social permitem identificar os principais eventos (mais de 20 tipos diferentes de eventos são fornecidos pelo GitHub) executados por um usuário, bem como, capturar a quantidade de *commits* realizados no projeto. Com base no tempo potencial (*date*) em que o projeto recebeu contribuições por meio de *commits*, introduzimos três características de engajamento que chamamos de ***atividade total do projeto***, ***contribuição diária*** e ***variação na periodicidade*** (encontram-

se formalizadas na Seção 5.4). Além disso, recuperamos as preferências do usuário por repositórios a partir das informações sobre os eventos nos repositórios, dado pelo quantitativo de eventos: *PushEvent*, *IssueEvent*, *ForkEvent*, *PullRequestEvent* e *WatchEvent*.

## 5.4 Atributos Derivados e Métricas

A partir dos dados disponibilizados, analisamos as atualizações dos projetos, por meio da periodicidade de atividade dos repositórios (data) e recebimento de contribuições diárias (*commits*). Além disso, propomos novos atributos derivados a partir das características recuperadas na plataforma GitHub. A seguir, apresentamos as métricas geradas de acordo com as ações dos usuários na plataforma. Na Tabela 5.1, exibimos todas as características descritas nesta seção.

No GitHub, podemos analisar o comportamento dos projetos por meio das atualizações visualizadas em sua linha do tempo. Com isso, torna-se possível criar métricas para acompanharmos a atividade do projeto na plataforma. De acordo com os autores Ponciano et al. [83], métricas de engajamento são medidas de interação e envolvimento do trabalhador com o projeto. Verificamos que o engajamento aplicado aos trabalhadores para computação por humanos poderia ser ajustado para analisar a atividade dos projetos utilizando as contribuições geradas pelos colaboradores na plataforma GitHub, conforme trabalho publicado pelos autores Cerqueira et al. [23].

Em um ambiente colaborativo, um projeto pode receber atualizações do desenvolvedor a qualquer momento. Este usuário pode ter o interesse de contribuir inicialmente para um projeto de curto prazo e não ter interesse em contribuir no longo prazo, por exemplo. As métricas propostas neste trabalho buscam analisar no detalhe a atividade dos projetos por meio dos *commits* (atualizações) que os contribuidores realizam ao longo do tempo, através da característica *date*. Todo projeto tem sua data de criação e, para cada *commit*, tem-se a data de atualização do projeto. Portanto, um dia ativo do projeto pode ser a data de criação, a data da última atualização ou qualquer outra a qual o projeto tenha recebido *commit*.

A Figura 5.2 mostra os conceitos utilizados na definição das métricas derivadas na linha do tempo do projeto: dias ativos do projeto, quantidade de *commits* em um dia ativo, data de

início do projeto, data da última atualização do projeto e o tempo decorrido entre cada par de dias ativos.


Figura 5.2: Estrutura de um projeto na linha do tempo.

Um dia ativo de um repositório $p$ é o dia em que este projeto recebeu pelo menos um *commit*. Nós definimos a Atividade Diária do Projeto - $AD_p$ - como um conjunto de datas em que o projeto esteve ativo (na Figura 5.2 o projeto recebeu *commits* em 28-05-2018, 18-06-2018, ..., 14-01-2019). Então, a quantidade de dias em que um projeto está ativo é dado por $|AD_p|$. O tempo que o projeto tem recebido contribuições é o número de dias decorridos entre a data em que o projeto foi criado e a data da última atualização do projeto (i.e., $max(AD_p)$). A seguir, apresentamos as métricas geradas a partir da definição de um dia ativo. Essas métricas enquadram-se no objetivo de analisar o nível de contribuição e duração das contribuições dos colaboradores nos projetos.

- A **Atividade Total do Projeto** - $ADR_p$ - permite analisar os dias ativos do projeto, desde o dia de criação até a sua última atualização, verificando-se a taxa de atividade de cada projeto no GitHub durante o período em que ele recebeu contribuições. Ou seja, é o número de dias em que um projeto recebeu contribuições em relação à diferença entre o último dia ativo do projeto (i.e. $max(AD_p)$) e o primeiro dia ativo do projeto (i.e. $min(AD_p)$). Este valor é calculado como indicado na Equação 5.2. Quanto mais próximo de 1, mais ativo é o projeto.

$$ADR_p = \frac{|AD_p|}{(max(AD_p) - min(AD_p)) + 1} \in (0, 1] \tag{5.2}$$

Por exemplo, na Figura 5.2, a quantidade de dias ativos é igual a 12 (quantidade de barras do gráfico, que indica a quantidade de *commits* diário). Vemos que o projeto foi criado em 07-05-2018 e seu último *commit* foi no dia 25-02-2019, então temos 294 dias de projeto. Então, aplicando a Equação 5.2 temos que $ADR_p = 12/295 = 0,04$ dias de atividade do projeto, ou seja, o projeto teve atualizações em 4% dos dias desde sua criação.

- A **Contribuição Diária do Projeto** - $DC_p$ - demonstrada pela a Equação 5.3, é a soma dos *commits* de todos os dias ativos contidos no conjunto $AD_p$ para o projeto em relação ao número de dias em que o projeto esteve ativo. Quanto maior a média, mais contribuições foram recebidas pelo projeto em cada dia ativo. Os *commits* realizados diariamente fornecem uma visão da extensão da contribuição diária nos projetos, onde definimos Número de *Commits* - $NMC_p(AD_p)$ - como o total de *commits* que o projeto recebeu em cada dia que permaneceu ativo.

$$DC_p = \frac{NC_p(AD_p)}{|AD_p|} \tag{5.3}$$

Ainda observando a Figura 5.2, temos os seguintes *commits* para cada dia: dia 1 (28-05-2018): 7, dia 2 (29-05-2018): 7, dia 3 (18-06-2018): 2, dia 4 (19-06-2018): 4, dia (20-06-2018)5: 3, dia 6 (21-08-2018): 3, dia 7 (12-09-2018): 3, dia 8 (03-12-2018): 1, dia 9 (30-12-2018): 9, dia 10 (12-01-2019): 2, dia 11 (13-01-2019): 2, dia 12 (14-01-2019): 3. Então, $DC_p = 46/12 = 3,83$.

- A **Variação na Periodicidade do Projeto** - $VP_p$ - captura como ocorre a periodicidade dos retornos de contribuição, ou seja, o período entre os dias que o projeto recebeu contribuições, podendo ser um retorno longo (demora para as atualizações) ou curto (brevidade nas atualizações). A $VP_p$ é o desvio padrão dos tempos decorridos entre cada par de dias ativos consecutivos. Define-se $E_p$ como o conjunto do número de dias decorridos entre cada dois dias ativos sequenciais. Quando $VP_p = 1$, o projeto apresenta a variação contínua no tempo decorrido entre cada par de dias ativos, o que indica que o projeto é atualizado diariamente, com perfeita periodicidade ($sd = 0$,

onde $sd$ indica a função de desvio padrão). Por outro lado, quanto maior a variação de $VP_p$, menos previsível será a periodicidade com que seus contribuidores atualizam o projeto. A variação é calculada de acordo com a Equação 5.4.

$$VP_p = sd(E_p) \tag{5.4}$$

Na Figura 5.2, sabemos que a quantidade de dias ativos do projeto são 12 e tem o total de 294 dias de projeto, temos $294 - 12 = 282$ dias de inatividade do projeto. Temos que entre o segundo dia ativo e o terceiro dia ativo, foram 20 dias inativo; do segundo dia ativo até o sexto dia ativo, foram 61 dias inativo; do sexto dia ativo até o sétimo dia ativo, foram 20 dias inativos; do oitavo dia ativo até o nono dia ativo, foram 92 dias inativos; do décimo dia ativo até o décimo primeiro dia ativo, foram 19 dias inativos. Assim, podemos dizer que $VP_p = sd(20, 61, 20, \ldots, 19) = 21.97$, mostrando ser um projeto que não mantém uma constância em suas atualizações.

## 5.5 Métodos de Classificação

A meta dos modelos de recomendação de projetos é prever os próximos projetos que o usuário teria interesse em contribuir. Portanto, precisamos definir como as técnicas de aprendizagem de máquina e recomendação selecionadas irão ser aplicadas ao conjunto de características selecionadas, tendo como classe o *fork* (não fazer *fork*- 0 ou fazer *fork*- 1). Nesta seção apresentamos as técnicas selecionadas aos dados coletados e os algoritmos utilizados para processamento dos dados pertencem ao pacote de aprendizado de máquina na linguagem de programação Python 2.7.14, conhecido como *Scikit-Learn*[3].

### 5.5.1 Filtragem Baseada em Conteúdo

Lembrando que os sistemas de recomendação baseados em conteúdo recomendam itens com características semelhantes aos de um determinado usuário no passado e que a similaridade

[3]https://scikit-learn.org/

Tabela 5.1: Características utilizadas pelos métodos de recomendação. Característica (CRT) Explícita (EX) ou Implícita (IM)

| **Característica** | **Grupo** | **Descrição** | **CRT** |
|---|---|---|---|
| events | ENG | Somatório dos eventos *IssueEvent*, *CommitEvent*, *PullRequestEvent* e *WatchEvent* por projeto | IM |
| year | ME | Ano de criação do projeto | EX |
| watchers | ATR | Quantidade de observadores em uma projeto | EX |
| followers | ATR | Quantidade de seguidores de um usuário | EX |
| language | ATR | Linguagem de programação principal do projeto | EX |
| commits | VIS | Quantidade de atualizações de um projeto | IM |
| weight_language | ATR | Peso da linguagem principal utilizada pelo usuário | IM |
| issues | VIS | Quantidade de assuntos tratados no projeto | IM |
| mean_commits | ME | Média de atualizações do projeto | IM |
| VP | ENG | Variação a periodicidade do projeto | IM |
| DC | ENG | Contribuição diária do projeto | IM |
| PR | VIS | Quantidade de requisições recebidas pelo projeto | IM |
| ADR | ENG | Atividade do projeto | IM |
| mean_watchers | ME | Média de observadores por projeto | IM |
| mean_pr | ME | Média de requisições por projeto | IM |
| max_issues | ME | Máximo de assuntos por projeto | IM |
| std_commits | ME | Desvio padrão das atualizações por projeto | IM |
| std_watchers | ME | Desvio padrão dos observadores por projeto | IM |
| max_pr | ME | Máximo de requisições por projeto | IM |
| max_commits | ME | Máximo de atualizações por projeto | IM |
| max_watchers | ME | Máximo de observadores por projeto | IM |
| mean_issues | ME | Média de assuntos por projeto | IM |
| std_pr | ME | Desvio padrão das requisições por projeto | IM |

dos itens é calculada com base nos recursos associados aos itens comparados, identificamos apenas 05 (cinco) características semelhantes entre usuários e projetos, dentre todas as características apresentadas na Tabela 5.1, para construção do vetor de características. A combinação das características do perfil do usuário, em que suas preferências e interesses são armazenados, com as características do perfil dos projetos, possibilita a recomendação ao usuário de novos projetos candidatos para contribuição [22]. As características selecionadas foram quantidade de seguidores (para usuário) ou quantidade de observadores (para projetos), quantidade de *commits* (para usuários e projetos), quantidade de *isues* (para usuários e projetos) quantidade de *pull-requests* (PR) (para usuários e projetos) e código da linguagem de programação (LP) (para usuários e projetos). As características de *followers* (seguidores) e *watchers* (observadores), por terem conceitos similares de popularidade, foram tratadas como semelhantes para usuários e projetos, mas são tratadas por nomes diferentes.

A partir dessas cinco características, definimos os perfis de usuário e projeto, e calculamos a similaridade entre eles. Por fim, os projetos mais populares do top-$N$ foram recomendados. O método consistiu nas seguintes etapas:

1. Representamos os usuários como um vetor contendo cinco características (quantidade de *commits* realizados pelo usuário, quantidade de *pull-requests* realizados pelo usuário, quantidade de *issues* que o usuário participou na plataforma (por exemplo, criar ou comentar ou ter sido marcado como responsável), quantidade de seguidores que o usuário possui na plataforma, código da principal linguagem de programação utilizada pelo usuário). O perfil do usuário foi definido da seguinte forma:

$$\vec{u} = (|\text{commits}|, |\text{PR}|, |\text{issues}|, |\text{watchers}|, \text{LP})$$

2. Representamos os projetos como um vetor contendo cinco características (quantidade de *commits* que o projeto recebeu, quantidade de *pull-requests* que o projeto recebeu, quantidade de *issues* sobre o projeto, quantidade de observadores que o projeto possui na plataforma, código da principal linguagem de programação do projeto). O perfil do projeto foi definido da seguinte forma:

$$\vec{i} = (|\text{commits}|, |\text{PR}|, |\text{issues}|, |\text{followers}|, \text{LP})$$

3. Considerando $m$ como a dimensão dos vetores, o perfil do usuário como vetor $\vec{u}$, e o do projeto como vetor $\vec{i}$, usamos a função de similaridade de cosseno (Equação 5.5 apresentada na Seção 5.6) para calcular a proximidade entre usuários e projetos.

4. Os itens top-$N$ para o usuário $u$ são calculados de acordo com a Equação 5.9 apresentada na Seção 5.6, verificando a similaridade entre $\vec{u}$ e $\vec{i}$, e retornando os $N$ elementos com similaridades mais altas.

### 5.5.2 FC Baseada em Vizinhança - Item-KNN

A abordagem dos vizinhos mais próximos é uma das mais comuns para filtragem colaborativa e, portanto, também adequada para projetar um sistema de recomendação. Esta abordagem funciona armazenando registros de treinamento e os utiliza para predizer a classificação do alvo desconhecido. Este modelo guarda em memória todo o conjunto de treinamento e, dado um ponto para ser classificado, ele encontra os $k$ pontos mais próximos (vizinhos mais próximos) dos registros de treinamento. Consequentemente, o modelo atribui a classificação do alvo de acordo com a classificação dos alvos de seus vizinhos mais próximos. A idéia principal é que, se um registro é determinado em uma vizinhança como uma classificação do alvo predominante, então o registro provavelmente pertence à mesma classificação.

Dada uma consulta $q$ cuja classe $l$ queremos conhecer e um conjunto de treinamento $X = x_1, l_1, \ldots, x_n, l_n$, em que $x_n$ é o $n$-th e $l_n$ é o alvo da classe, o classificador kNN determinará um subconjunto $Y = y_1, l_1, \ldots, y_k, l_k$ onde $Y \subseteq X$ e $\sum_1 \mathrm{d}(q, y_k)$ é mínimo. $Y$ contém os $k$ pontos em $X$ que estão mais próximos do ponto de consulta $q$. Em seguida, o rótulo de classe de $q$ é $l = f(l_1 \ldots l_k)$ [92].

As técnicas de filtragem colaborativa contam com a similaridade usuário-usuário ou item-item para realizar a recomendação. Para isso, um sistema de filtragem colaborativa precisa definir duas entidades: itens e usuários. Os usuários $(u, v) \in U$, onde $U$ é o conjunto de usuários e $i \in I$, onde $I$ é o conjunto de itens. Cada usuário $u$ é associado a um subconjunto $I_u \subset I$ de itens avaliados por $u$. Da mesma forma, cada item $i$ é associado a um subconjunto $U_i \subset U$ de usuários que avaliaram o item $i$.

Por exemplo, como mostra a Figura 5.3, o usuário $u_1$ gostaria de encontrar outros projetos, além dos projetos $p_1$ e $p_2$, para contribuir. Ele sabe que o $u_2$ tem gosto semelhante ao

dele, pois eles dois contribuíram para o projeto $p_2$. Então, o usuário $u_1$ poderia perguntar ao usuário $u_2$ sobre o projeto $p_3$. Por outro lado, o usuário $u_1$ descobre que ele e o usuário $u_3$ têm gostos diferentes, pois eles não contribuem em projetos semelhantes e, neste caso, descarta a opinião do usuário $u_3$. Assim, podemos inferir que $u_1$ tem gosto mais similar ao $u_2$ do que ao usuário $u_3$ e, quanto maior for a correspondência de interesses de projetos semelhantes entre usuários, maior será a confiança que o algoritmo terá de recomendar projetos semelhantes para esses usuários. Por fim, ao contrário dos sistemas baseados em conteúdo, os de filtragem colaborativa podem recomendar itens com conteúdo diferente, desde que outros usuários já tenham mostrado interesse por esses itens.

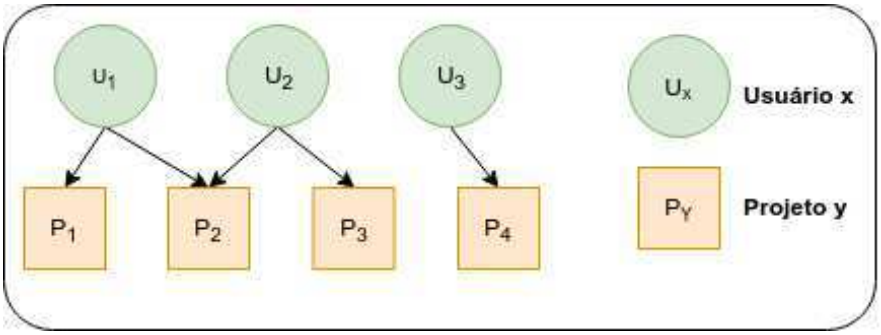


Figura 5.3: Exemplo de relacionamentos, por exemplo, entre usuários e projetos

Os classificadores baseado em vizinhança estão entre os mais simples de todos os algoritmos de aprendizado de máquina e têm gerado bons resultados de precisão em vários estudos e, por ser uma abordagem intuitiva e direta, são bastante flexíveis para melhorias [6]. No nosso trabalho, utilizamos o Item-KNN com todas as características de usuário e projeto apresentados na Tabela 5.1.

### 5.5.3 Árvore de Decisão

Árvores de decisão são classificadores em um atributo (ou classe) de destino na forma de uma estrutura em árvore. Os itens a serem classificados contêm o valor de atributo e o atributo de destino. Os nós da árvore podem ser: i) nós de decisão, onde um único valor de atributo é testado para determinar a qual ramificação da subárvore se aplica; ou ii) nós de folha que indicam o valor do atributo de destino. Entretanto, uma única árvore não é forte o suficiente para gerar melhores resultados na predição. Geralmente, utiliza-se modelos compostos por conjuntos de árvores, que resume a previsão de várias árvores juntas em um resultado final.

Neste trabalho, utilizamos os dois algoritmos mais utilizados na atualidade que aplicam o conceito de árvore de decisão: *Random-Forest* [45] e *extreme gradient boosting* (XGBoost) [27]. Eles mapeiam relacionamentos não lineares de forma satisfatória e podem ser adaptados para resolver diferentes tipos de problemas.

O *Random Forest* é um algoritmo de aprendizagem supervisionada, que cria uma floresta composta de muitas árvores de decisão de forma aleatória. O algoritmo induz o uso de florestas aleatórias, desenvolvido por Cutler et al. [31], sendo *Random Forest* sua marca registrada. O termo *Random Forest* surgiu a partir do estudo realizado e proposto, pela primeira vez, por Ho [50] em 1995. O método combina a idéia de *bagging* de Cutler et al. [31] e a seleção aleatória de amostras, introduzidos por Ho [50], a fim de construir uma combinação dos modelos de aprendizado que objetiva melhorar o resultado geral.

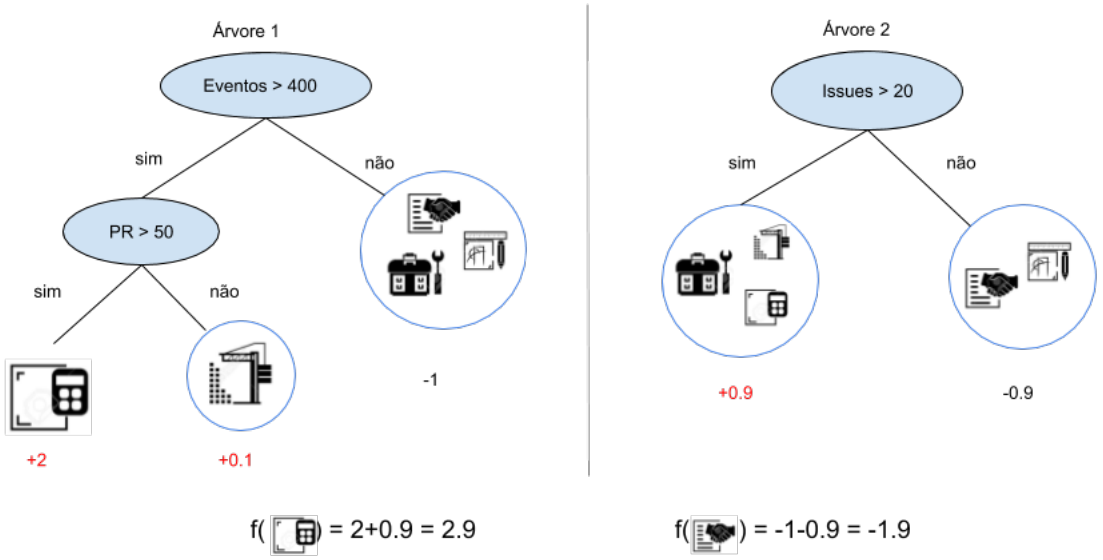


Figura 5.4: Modelo de conjunto de árvores. A previsão final para um determinado exemplo é a soma das previsões de cada árvore.

No exemplo da Figura 5.4, podemos visualizar duas árvores de decisão que compõem uma floresta aleatória, onde cada árvore depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição para todas as árvores da floresta. Essa coleção de classificadores estruturados em árvore são vetores aleatórios distribuídos de forma idêntica e independentes e que cada árvore emite um voto unitário para a classe na entrada [17]. Ou seja, para a Árvore 1, o atributo Eventos para valores maiores que 400 tem a quantidade de PR maiores que 50 para projetos que tratam de assuntos financeiros. O exemplo é classificado como Sim, ou seja, quando existir Eventos maiores que 400 e PR maiores

que 50 a classe é marcado com *fork* igual a 1 (ou melhor, uma probabilidade para *fork* ser próximo de 0 ou 1). Assim são feitas diferentes combinações para as diferentes árvores criadas. As pontuações de previsão de cada árvore são individuais e, ao final, é aplicado a média das pontuações, sendo resumida para uma pontuação final. Assim, o algoritmo combina os resultados para obter uma predição com maior acurácia.

Se pensarmos em florestas aleatórias e árvores potencializadas, percebemos que as duas têm modelos semelhantes: são árvores de decisão que compõem uma floresta aleatória. A diferença surge de acordo com a forma que treinamos esses modelos. Essa é a proposta do *XGBoost*. Este modelo é uma implementação bem conhecida de código aberto e eficiente do algoritmo baseado em árvores de decisão. Ele foi apresentado em 2016 por Chen and Guestrin [25] na Conferência SIGKDD[4] e, desde então, tem sido bastante utilizado pelos profissionais da área.

Assim como o *Random Forest*, o *Extreme Gradient Boosting* é um algoritmo de aprendizado supervisionado que tenta prever com precisão uma variável de destino. Entretanto, ele busca otimizar as árvores de decisão de ponta a ponta, de forma sequencial. Cada nó da árvore é uma pergunta sobre as variáveis e cada folha é a previsão do alvo. Como mostra a Figura 5.5, o *Gradient Boosting* é a combinação do resultado de muitos classificadores (ou regressores) fracos, se combinando para formar uma espécie de comitê forte de decisão. Embora não haja restrições quanto a estes classificadores, é usual a utilização de árvores de decisão. Assim, o fator mais importante por trás do sucesso do *XGBoost* é a combinação de resultados aliado à escalabilidade em todos os cenários, possibilitando a execução dez vezes mais rápida que outras soluções baseadas em árvore de decisão [25].

Como mostra a Figura 5.5, basicamente, para uma determinada estrutura de árvore, recupera-se o valor do gradiente estatístico de $gi$ e $hi$ para as folhas às quais pertencem, soma-se as estatísticas e verifica-se o quão acurado são os resultados. Assim, analisamos o quão assertiva é a árvore de decisão, levando em consideração a complexidade do modelo.

O *XGBoost* tem se mostrado mais robusto no manuseio quando da variedade de tipos de dados, relacionamentos e distribuições. Outrossim, apresenta um grande número de hiperparâmetros que podem ser ajustados para um cenário mais apropriado. Essa flexibilidade torna o *XGBoost* uma escolha consistente para problemas de regressão e classificação. O

[4]https://www.kdd.org/kdd2016/

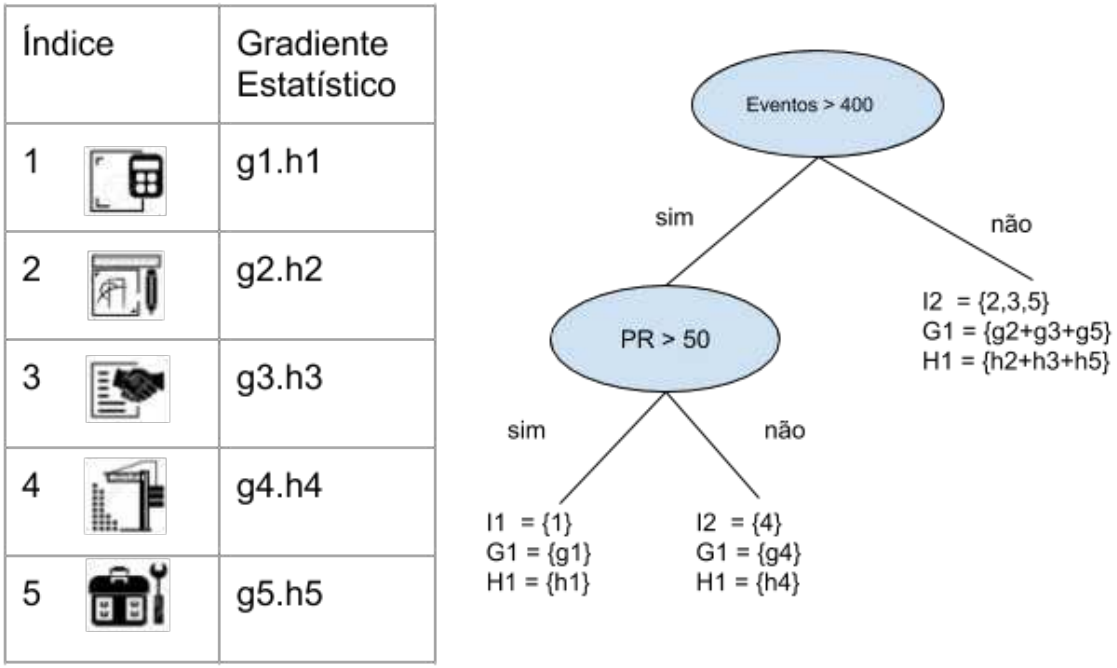


Figura 5.5: A estrutura exibe a soma do gradiente estatístico para cada nó folha e define o *score*.

número máximo de níveis em cada árvore de decisão utilizado em nosso estudo foi de 05 (cinco).

### 5.5.4 Máquina de Fatoração

A máquina de fatoração (MF) é um modelo supervisionado poderoso que amplia significativamente a fatoração da matriz. Ela é adequada para dados transacionais de cardinalidade muito alta e escassamente observados, como observado em algumas características do GitHub (por exemplo, as linguagens de programação). Ou seja, nenhum projeto contém todas as linguagens de programação disponibilizadas na plataforma, como mostra a Tabela5.2.

Podemos visualizar, que todas as características são representadas por uma coluna e são chamadas de variáveis independentes. Além disso, temos a classe *fork*, que também é conhecida por variável dependente das demais [89]. Utilizamos as 22 características definidas neste trabalho e as 20 principais linguagens de programação demonstradas no Capítulo 4. Podemos ver que para cada característica é criada uma coluna, por exemplo, o projeto $p_1$ tem 45 *commits*, 4 *PR* e 20 *eventos*. No caso das características conhecidas como categóricas (são as características que não possuem valores quantitativos, ou seja, representam uma classificação dos projetos), por exemplo, o nome da linguagem de programação, temos que

Tabela 5.2: Modelo de Máquina de Fatoração

| | **vetor de características** | | | | | | **alvo** | |
|---|---|---|---|---|---|---|---|---|
| | | | | x | | | y | |
| $p_1$ | 45 | 4 | 2 | 90 | 1 | 0 | 0 | $y_1$ |
| $p_2$ | 498 | 27 | 15 | 780 | 0 | 1 | 1 | $y_2$ |
| $p_3$ | 321 | 109 | 54 | 1020 | 1 | 0 | 1 | $y_3$ |
| $p_4$ | 56 | 3 | 19 | 43 | 0 | 0 | o | $y_4$ |
| $p_5$ | 2098 | 98 | 67 | 4987 | 0 | 1 | 1 | $y_5$ |
| $p_6$ | 45 | 0 | 2 | 104 | 0 | 1 | 0 | $y_6$ |
| $p_7$ | 789 | 10 | 23 | 456 | 0 | 1 | 1 | $y_7$ |
| | *commits* | $PR$ | *issues* | *eventos* | $Java$ | $Python$ | | |
| | **usuários** | | | | **itens** | | | |

para cada LP é criada uma nova coluna e adicionado valores 0 ou 1 para os projetos que utilizam essa LP ou não. Por exemplo, temos que o projeto $p_1$ utiliza apenas a linguagem de programação Java, já o projeto $p_45$ utiliza a linguagem de programação Python. Essa matriz é potencialmente muito grande, porque pode haver milhões de usuários e projetos. Além disso, é escassamente observada, porque geralmente apenas uma fração muito pequena das classificações históricas está disponível. O objetivo é prever qual classificação um usuário poderia fornecer hipoteticamente a um conjunto de projetos e, em seguida, recomendar itens que o usuário provavelmente tenha interesse.

## 5.6 Métricas

Para o modelo de filtragem baseado em conteúdo, primeiramente, recuperamos as preferências do usuário com base nas características dos seus projetos disponíveis na plataforma GitHub. Em seguida, usando os perfis do usuário e projeto, calculamos a similaridade (ver Equação 5.5) entre o perfil do usuário e todos os os projetos do conjunto de dados. Finalmente, calculamos a acurácia do modelo (ver Equação 5.6, Equação 5.7, Equação 5.8) e criamos uma lista decrescente contendo os projetos mais similares (quanto mais próximo de 1, mais similares são os projetos) ao perfil do usuário e, por fim, recomendamos o top-N

projetos (ver Equação 5.9).

Para os demais modelos, primeiramente, recuperamos as características dos usuários e projetos. Em seguida, recuperamos o valor da classe *fork* (positivo - 1 ou negativo - 0), gerando o conjunto de dados de treinamento. Aplicamos os modelos ao conjunto de dados de treinamento para o aprendizado dos dados e logo em seguida realizamos a predição para os projetos candidatos, ou seja, o foco está na criação da lista de predição de classificações para projetos não classificados (neste caso, o valor binário de *fork*) de um usuário. Esta lista contém, em ordem descrescente, a predição da classificação dos projetos para cada usuário. Finalmente, calculamos a acurácia do modelo a partir do conjunto de teste (ver Equação 5.6, Equação 5.7, Equação 5.8), validando os projetos da lista de predição para top-N e realizando a recomendação (ver Equação 5.9).

A métrica de similaridade é tipicamente empregada para calcular uma pontuação (similaridade) entre o perfil do usuário e cada um dos projetos candidatos (perfil de itens) do conjunto de dados. Esta pontuação é calculada de acordo com a abordagem adotada, que pode diferir em termos de construção de perfil e métrica de similaridade. Nós utilizamos a função de similaridade do Cosseno (Equação 5.5) para calcular a similaridade entre usuários e projetos [7].

$$sim(\vec{u}, \vec{i}) = \frac{\sum_{k=1}^{m} u_k \cdot i_k}{\sqrt{\sum_{k=1}^{m} (u_k)^2} \cdot \sqrt{\sum_{k=1}^{m} (i_k)^2}} \tag{5.5}$$

A similaridade do cosseno retorna valores entre 0 e 1. O valor 1 indica que ambos os vetores têm máxima semelhança, portanto, quanto mais próximo de 1, mais forte é a relação entre os usuários e projetos e, quanto mais próximo de 0, mais insignificante é essa relação.

Tipicamente, *precision*, *recall* and *f-measure* são utilizadas para medir a qualidade das recomendações, todos para o $k$ primeiras recomendações. Essas métricas são baseadas no arcabouço conceitual proposto por Resnick and Varian [91]. *Precision* (Equação 5.6) é definida como a fração do conjunto de projetos recomendados que são relevantes, *recall* (Equação 5.7) é a fração do conjunto de projetos do usuário e que foi de fato recomendado e *f-measure* é a média harmônica entre essas duas métricas.

Tendo que $Y$ é o conjunto relevante de projetos para o usuário $u$ ($Y = Y_u$). Seja $C$ o conjunto ordenado de recomendações geradas pelo método que está sendo avaliado, $C^k$ é o

conjunto top $k$ elementos em $C$ e $C_i$ o $i_{th}$ elemento em $C$.

*Precision* nas primeiras $k$ posições da classificação, $precision@k$, é definida como:

$$precision@k(C, Y) = \frac{|\{C^k\} \cap \{Y\}|}{|\{C^k\}|} \tag{5.6}$$

*Recall*, por sua vez, é definida como:

$$recall@k(C, Y) = \frac{|\{C^k\} \cap \{Y\}|}{|\{Y\}|} \tag{5.7}$$

Por fim, *f-measure* é definida como:

$$f - measure = 2 * \frac{(precision * recall)}{precision + recall} \tag{5.8}$$

Os projetos top-$N$ para o usuário $u \in U$ são calculados como apresentados na Equação 5.9. Independentemente da abordagem utilizada, os diagramas top-$N$ com pontuações mais altas serão recomendados ao usuário. O valor de $N$, utilizado nesta pesquisa, variou desde as recomendações mais próximas do interesse do usuário, como top-3, até top-100. Decidimos apresentar resultados até top-100 para entender melhor o alcance e posição das recomendações.

$$\text{top-}N(\vec{u}) := \underset{\vec{i} \in I}{\overset{n}{\text{argmax}}}\ sim(\vec{u}, \vec{i}) \tag{5.9}$$

## 5.7 Considerações Finais

Neste capítulo, discutimos como foi realizado o pré-processamento dos dados, a partir de duas amostragens propostas: balanceada e negativa. Contextualizamos os conceitos das características sociais e criamos outras características derivadas a partir da análise da linha do tempo dos projetos quanto ao engajamento das contribuições aplicadas nos projetos. Apresentamos também a aplicação dos algoritmos selecionados no contexto de recomendação de projetos e, por fim, apresentamos as métricas a serem utilizadas para validação dos resultados.

# Capítulo 6

# Análise e Resultados

Neste capítulo, apresentamos a análise dos principais resultados do nosso experimento e discutimos nossas descobertas sob a perspectiva das questões de pesquisa. Conforme declarado anteriormente no Capítulo 1, o experimento foi projetado para abordar as seguintes questões de pesquisa (QP):

- QP1 - Como atuam os sistemas de recomendação no domínio de projetos do GitHub em termos de fatores de qualidade das recomendações?
- QP2 - Quais são as características, dentre os conjuntos de dados de atratividade, visibilidade, engajamento e medida das estatísticas, que se mostram mais relevantes em termos de fatores de qualidade das recomendações de projetos?
- QP3 - Qual o efeito nos fatores de qualidade das recomendações, quando aplicada uma nova metodologia de teste a partir da amostragem negativa?

Essas perguntas são abordadas utilizando todas as características analisadas (Seção 5.3 e Seção 5.4), utilizando dois conjuntos de dados (Seção 5.2) e comparando as técnicas de recomendação selecionadas (Seção 5.5), de acordo com a metodologia de avaliação proposta (Seção 5.6).

## 6.1 Validação da Qualidade da Recomendação

Nesta seção, para respondermos QP1, investigamos como os métodos de recomendação propostos atuam no domínio dos projetos e aplicamos as métricas (Seção 5.6) para avaliar a

qualidade das recomendações. Inicialmente, aplicamos as características definidas na Seção 5.3 e Seção 5.4, exibidas pela Tabela 5.1. Os fatores de qualidade representam a métricas aplicadas nos experimentos: (*precision*, *recall*) e *f-measure*. Para $top - N$, utilizamos $N$ recomendações igual a $top - 3$, $top - 5$, $top - 10$, $top - 50$ e $top - 100$.

Primeiramente, aplicamos as características ao modelo de **Filtragem Baseada em Conteúdo - FBC**, descritas na Seção 5.5.1. Por ser uma classificação não supervisionada, o rótulo não é utilizado [6]. Assim, organizamos adequadamente as características utilizadas, selecionando apenas aquelas que são comuns entre os perfis do usuário e projeto. Por exemplo, para a característica de popularidade, temos que o projeto contém uma determinada quantidade de *watchers* e o usuário contém uma determinada quantidade de *followers*. Utilizamos a semelhança entre elas e definimos as 05 (cinco) características similares entre usuário e projeto: *followers*, *watchers*, *issues*, *commits*, PR e *weight_language*, como mostra a Tabela 6.1. Para realização do experimento, coletamos as características de acordo com as duas amostragens: balanceado e negativa.

Tabela 6.1: Amostra das Características Semelhantes entre Usuário e Projeto

| | *watchers/followers* | *commits* | *weight_language* | *issues* | *PR* |
|---|---|---|---|---|---|
| $u_1$ | 993 | 13038 | 0.74 | 500 | 1390 |
| $p1_{u1}$ | 815 | 434 | 0.2092 | 449 | 149 |
| $u_2$ | 268 | 668 | 0.4 | 521 | 2715 |
| $p1_{u2}$ | 2315 | 668 | 0.0631 | 125 | 18 |

Segundo, além das características semelhantes entre usuário e projeto, selecionadas as demais características descritas na Tabela 5.1 e aplicamos aos métodos de classificação supervisionada apresentados na Seção 2.2 (**Random-Forest**, **XGBoost**, **MF** e **Item-KNN**). Utilizamos o pacote de aprendizado de máquina na linguagem de programação Python 2.7.14, conhecido como *Scikit-Learn*, para execução dos algoritmos selecionados. A abordagem mais comum à modelagem preditiva é selecionar um conjunto de variáveis que são consideradas preditivas para uma tarefa. Essa tarefa tem por objetivo aprender a dependência funcional das variáveis preditoras aos interesses do alvo. Ao longo deste trabalho, tivemos a tarefa de prever a decisão do alvo de contribuir ou não nos projetos disponibilizados na plataforma. Na Tabela 6.2, apresentamos uma pequena amostra dos valores definidos para

as características, exibindo os valores coletados na plataforma (*issues*, *commits*, *watchers*, ..., *pull-requests*), e que são preditivos para estimar a decisão pelo projeto ser *fork* ou não (1 - receber contribuições ou 0 - não receber contribuições) por algum usuário. Da mesma forma, utilizamos os dois tipos de amostragens: balanceado e negativa.

Tabela 6.2: Amostra das Características no Conjunto de Treino

| | *issues* | *commits* | *watchers* | ADR | ... | *PR* | Alvo (y) |
|---|---|---|---|---|---|---|---|
| $x_1$ | 252 | 46 | 2632 | 0.75 | ... | 12 | 1 |
| $x_2$ | 101 | 457 | 1804 | 0.17 | ... | 75 | 0 |
| $x_3$ | 91 | 150 | 1016 | 0.43 | ... | 27 | 0 |
| $x_4$ | 33 | 33 | 1710 | 0.07 | ... | 19 | 1 |

Em nossos experimentos, as métricas *precision* e *recall* são aplicadas para avaliar a qualidade da recomendação dos repositórios. A Tabela 6.3 exibe os resultados experimentais. Temos uma análise da qualidade da recomendação para $top-3$, $top-5$, $top-10$, $top-50$ e $top-100$. Entre todos os modelos, o XGBoost tem o melhor desempenho para *precision* porque fornece automaticamente estimativas de importância das características. Para *recall*, XGBoost e Random-Forest superam os demais modelos porque otimizam a métrica diretamente sobre as classificações.

É aparente que os modelos apresentam disparidades significativas de desempenho entre eles em termos de *precision* e *recall* para $top-N$. Por exemplo, na amostragem negativa e $top-10$, a métrica *precision* para o modelo não supervisionado FBC é cerca de 0,00017 e para o modelo supervisionado MF é cerca de 0,00011. Apesar dos modelos apresentarem formas de classificação diferentes e quantitativo de características diferentes, o aprendizado apresentou-se de forma semelhante. Esperávamos que o modelo MF apresentasse um melhor desempenho, pois modela todas as interações possíveis entre os valores no vetor de características utilizando interações fatoradas, mesmo sob alta esparsidade. Entretanto, o MF não apresentou resultados melhores que os demais modelos. Em relação à amostragem balanceada, os valores de *precision* para XGBoost e RF estão mais próximos, apresentando um desempenho de acerto de 0,00089 e 0,00099, respectivamente, para $top-10$, mas XGBoost apresenta resultados melhores em comparação à amostragem negativa. A Figura 6.1 compara o aprendizado dos modelos em relação à métrica *precision* aplicada as duas amostragem

selecionadas.

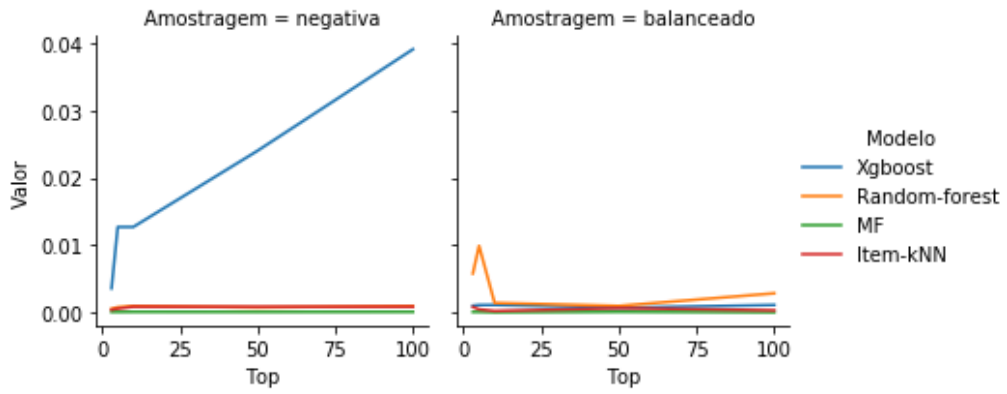


Figura 6.1: Resultados de *Precision* para Todos os Modelos

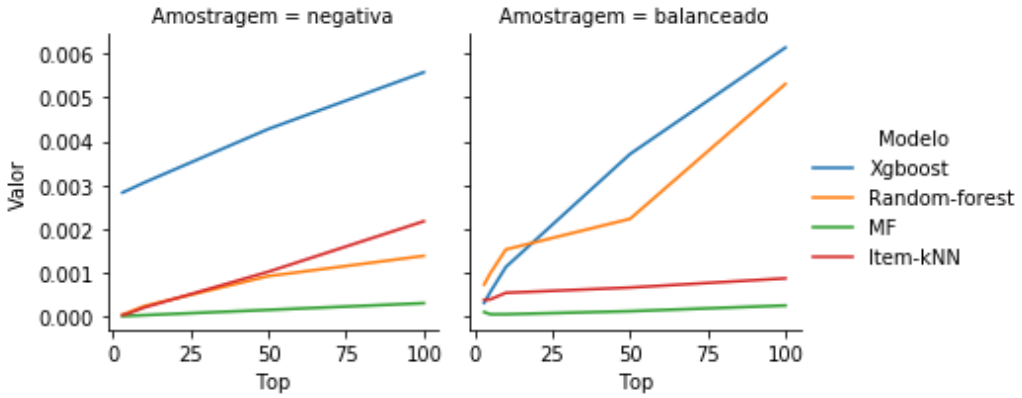


Figura 6.2: Resultados de *Recall* para Todos os Modelos

Para a métrica *recall*, como mostra a Figura 6.2, quando comparamos os modelos nas duas amostragens, vemos que Item-KNN apresenta resultados melhores que MF, mas isto não acontece para a métrica *precision*, onde os mesmos possuem resultados semelhantes. Entretanto, vemos que os melhores resultados apresentados são para Random-Forest e XG-Boost, que demonstram valores similares para $top - N$, para *recall*, podendo chegar até 0,00557 na recomendação de projetos candidatos. No geral, podemos verificar que o modelo *XGBoost* apresenta os melhores resultados para *recall* nas duas amostragens utilizadas, mas temos que para *precision*, na amostragem balanceada, XGBoost tem valores similares aos demais modelos.

Observando os resultados por meio da métrica *f-measure* e pela ótica das amostragens utilizadas, vemos que os resultados dos modelos apresentados na Tabela 6.3 e plotados na

Figura 6.3, comportaram-se de formas diferentes. Para o modelo XGBoost, este apresenta um melhor resultado de aprendizado quando aplicado à amostragem negativa. Ao contrário do modelo RF, que tem melhor resultado de aprendizado utilizando a amostragem balanceada. Podemos inferir que o modelo XGBoost apresenta melhor previsão quando aplicado a um ambiente mais próximo ao ambiente do GitHub, ao contrário do que acontece com o modelo RF [111]. Para os demais modelos, Máquina de Fatoração e Item-KNN, o aprendizado é semelhante, mas o modelo MF não conseguiu superar o clássico Item-KNN, nem o modelo não supervisionado FBC. Entretanto, em termos de valores, o modelo XGBoost apresenta melhores resultados desde *top-3* até *top-100*.

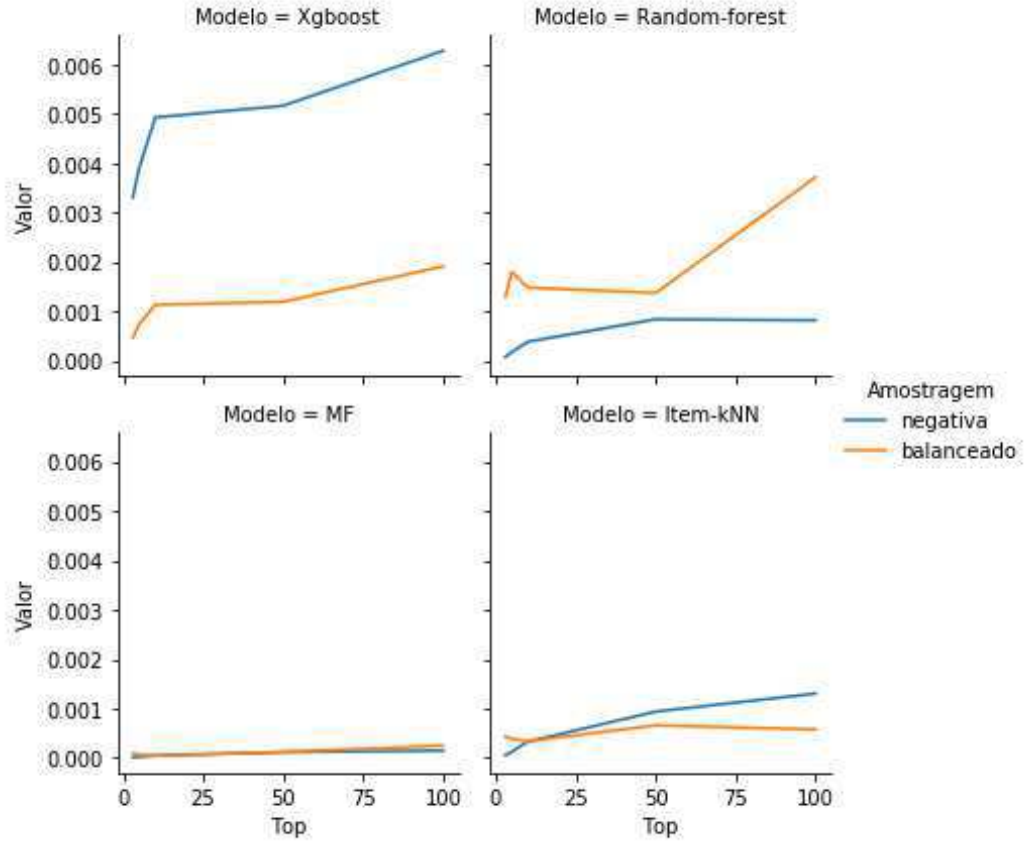


Figura 6.3: Comparação entre Modelos usando F-Measure

No geral, podemos verificar que (i) a abordagem Item-KNN apresenta um custo de classificação alto, embora simples e intuitiva, não mostrou bons resultados, estando um pouco melhor que MF e FBC; (ii) os modelos baseados em árvore de decisão são mais robustos ao lidar com uma variedade de tipos de dados, relacionamentos e distribuições, bem como com um grande conjunto de hiper-parâmetros que podem ser refinados e ajustados para um cená-

Tabela 6.3: *Precision*(P), *Recall*(R), *F-Measure*(F-M) para Amostragem Balanceada e Negativa

| | Dataset | Amostragem Balanceada | | | Amostragem Negativa | | |
|---|---|---|---|---|---|---|---|
| | | P | R | F-M | P | R | F-M |
| FBC | top-3 | 0.00014 | 0.00014 | 0.00014 | 0.00016 | 0.00010 | 0.00012 |
| | top-5 | 0.00015 | 0.00015 | 0.00016 | 0.00018 | 0.00017 | 0.00017 |
| | top-10 | 0.00014 | 0.00015 | 0.00016 | 0.00017 | 0.00019 | 0.00017 |
| | top-50 | 0.00016 | 0.00018 | 0.00017 | 0.00019 | 0.00018 | 0.00017 |
| | top-100 | 0.00019 | 0.00020 | 0.00019 | 0.00022 | 0.00022 | 0.00022 |
| Item-KNN | top-3 | 0.00072 | 0.00028 | 0.00052 | 0.00004 | 0.00003 | 0.00005 |
| | top-5 | 0.00049 | 0.00029 | 0.00043 | 0.00058 | 0.00007 | 0.00012 |
| | top-10 | 0.00024 | 0.00034 | 0.00033 | 0.00087 | 0.00020 | 0.00033 |
| | top-50 | 0.00066 | 0.00066 | 0.00066 | 0.00087 | 0.00102 | 0.00093 |
| | top-100 | 0.00034 | 0.00087 | 0.0.00048 | 0.00087 | 0.00217 | 0.00130 |
| RF | top-3 | 0.00587 | 0.00129 | 0.00066 | 0.00560 | 0.00004 | 0.00007 |
| | top-5 | 0.00099 | 0.00099 | 0.0018 | 0.00080 | 0.00009 | 0.00017 |
| | top-10 | 0.00099 | 0.00153 | 0.00148 | 0.00099 | 0.00230 | 0.00038 |
| | top-50 | 0.00099 | 0.00223 | 0.00137 | 0.00077 | 0.00092 | 0.00083 |
| | top-100 | 0.00286 | 0.00531 | 0.00371 | 0.00099 | 0.00138 | 0.00081 |
| XGBoost | top-3 | 0.00065 | 0.00031 | 0.00047 | 0.00359 | 0.00283 | 0.00331 |
| | top-5 | 0.00102 | 0.00084 | 0.00074 | 0.01270 | 0.00289 | 0.00391 |
| | top-10 | 0.00089 | 0.00101 | 0.00113 | 0.01270 | 0.00305 | 0.00492 |
| | top-50 | 0.00049 | 0.00371 | 0.00119 | 0.02401 | 0.00428 | 0.00516 |
| | top-100 | 0.00101 | 0.00501 | 0.00191 | 0.03906 | 0.00557 | 0.00631 |
| MF | top-3 | 0.00010 | 0.00010 | 0.00010 | 0.00010 | 0.00001 | 0.00001 |
| | top-5 | 0.00010 | 0.00005 | 0.00006 | 0.00010 | 0.00001 | 0.00002 |
| | top-10 | 0.00005 | 0.00005 | 0.00005 | 0.00011 | 0.00002 | 0.00004 |
| | top-50 | 0.00012 | 0.00012 | 0.00012 | 0.00010 | 0.00015 | 0.00012 |
| | top-100 | 0.00005 | 0.000025 | 0.00025 | 0.00011 | 0.00030 | 0.00015 |

rio mais apropriado. Por isso, sua aplicação segregou bem os projetos com base nos valores das características, identificando mais precisamente as variáveis que definem os melhores conjuntos homogêneos de projetos (que são heterogêneos entre eles) e, então, apresentando melhores resultados; (iii) embora o MF, otimizando a característica - linguagem de programação, não conseguiu superar os demais modelos, mesmo sabendo que as interações entre os valores podem ser estimadas sob alta esparsidade. Especialmente, sendo possível generalizar para interações não observadas. Em geral, pode-se observar que o XGBoost, além de ser o método de recomendação de maior precisão, teve um resultado geral melhor em termos de cobertura dos projetos.

Por fim, por utilizarmos um conjunto de dados pré-coletados e não realizarmos nenhuma interação on-line com os usuários, os modelos de classificação supervisionada obtiveram resultados melhores e aprenderam de acordo com os perfis dos usuários e projetos coletados. No que podemos concluir que o modelo RF aprende melhor para amostras que contém um equilíbrio das classes *fork* (amostragem balanceada). Enquanto que o modelo XGBoost tem melhor desempenho com o efeito do desequilíbrio das classes *fork* (amostragem negativa).

## 6.2 Efeito das Características no Aprendizado dos Modelos

Nesta seção, investigamos a importância das características para o aprendizado dos modelos. De acordo com as técnicas utilizadas e analisados os projetos mais recomendados, identificamos que estes mostram uma maior diferença entre si em termos de quantidade de observadores, quantidade de *commits*, linguagens de programação e quantidade de *pull-requests*. Portanto, vamos analisar todas as características e seus efeitos para a qualidade das recomendações.

Inicialmente, analisamos o aprendizado para as linguagens de programação e, como mostra a Figura 6.4, vemos que as linguagens de programação dos projetos mais recomendados estão bem próximas das linguagens de programação mais utilizadas pela plataforma GitHub (ver Figura 4.1). Mantendo-se *javascript*, *java*, *Ruby* e *Python* como as principais linguagens de programação dos projetos mais recomendados.

Como o modelo XGBoost apresentou melhores resultados para *precision* e *recall*, como demonstrado na Seção 6.1, decidimos analisar o efeito das características para este modelo.


Figura 6.4: Linguagens de Programação dos Projetos Mais Recomendados

Utilizamos as 22 características propostas em nossa metodologia. Selecionamos os *top-10* projetos mais recomendados no uso do modelo XGBoost. Como mostra a Tabela 6.4, eles têm como linguagens de programação predominantes Python e JavaScript. Estes projetos tratam, basicamente, de assuntos como *deep learning* e melhores experiências e práticas no uso da linguagem de programação. Uma outra observação é que alguns dos usuários, donos dos projetos, são organizações (*Keras*, *Gorhill* e *DkHamsing*). Identificamos que os projetos mais ativos (contém mais *commits* e *pull-request*) nem sempre estão entre os mais recomendados. Ou seja, os projetos mais atualizados da plataforma podem não ser os mais recomendados para os usuários.

Também verificamos que os projetos mais recomendados contém as *top-10* linguagens de programação, de acordo com a Figura 6.4 equivalem a 88.0% dos projetos recomendados e que os projetos com o maior número de observadores também não estão entre os projetos mais recomendados. Os *top-10* projetos listados na Tabela 6.4 tem em média 1.8 *watchers*, 9.8 *commits*, 8.6 *pull-request* e 1223.8 *issues*. Como podemos verificar, a popularidade de um projeto e sua atualização constante, não parecem ser os principais fatores para a recomendação de um projeto.

Decidimos realizar uma outra análise para o modelo XGBoost utilizando outras medidas

| **Projetos** | **Usuários** | **Quantidade de Recomendações** | **Linguagem de Programação Principal** |
|---|---|---|---|
| **thefuck** | NVBC | 18698 | Python |
| **big-list-of-naughty-strings** | Minimaxir | 17912 | Python |
| **hacker-scripts** | NARKOZ | 17816 | JavaScript |
| **clipboard.js** | ZenoRocha | 17770 | JavaScript |
| **dragula** | Bevacqua | 17734 | JavaScript |
| **uBlock** | Gorhill | 17716 | JavaScript |
| **keras** | Keras | 16988 | Python |
| **neural-style** | JcJohnson | 16946 | Lua |
| **open-source-ios-apps** | DkHamsing | 16940 | Python |
| **react-boilerplate** | React-Boilerplate | 16922 | JavaScript |

Tabela 6.4: Top-10 projetos recomendados utilizando XGBoost

de validação para identificarmos as características mais importantes para o modelo XGBoost. Analisamos três medidas típicas de importância, que estão listadas abaixo.

1. ***Weight***. O número de vezes que uma característica é usada para dividir os dados em todas as árvores.

2. ***Cover***. O número de vezes que uma característica é usada para dividir os dados em todas as árvores ponderadas pelo número de pontos de dados de treinamento que passam por essas divisões.

3. ***Gain***. A redução média de perda de treinamento obtida ao usar uma característica para divisão.

O *weight* é a opção padrão, exibida na Figura 6.5, que apresenta um gráfico de barras simples, representando a importância de cada característica para o modelo XGBoost. Analisando-as, vemos que a quantidade de eventos domina em relação às demais. Isto é um tanto curioso, pois esta característica é o somatório dos eventos: *PushEvent*, *IssueEvent*, *CommitEvent*, *PullRequestEvent* e *WatchEvent* por projeto.

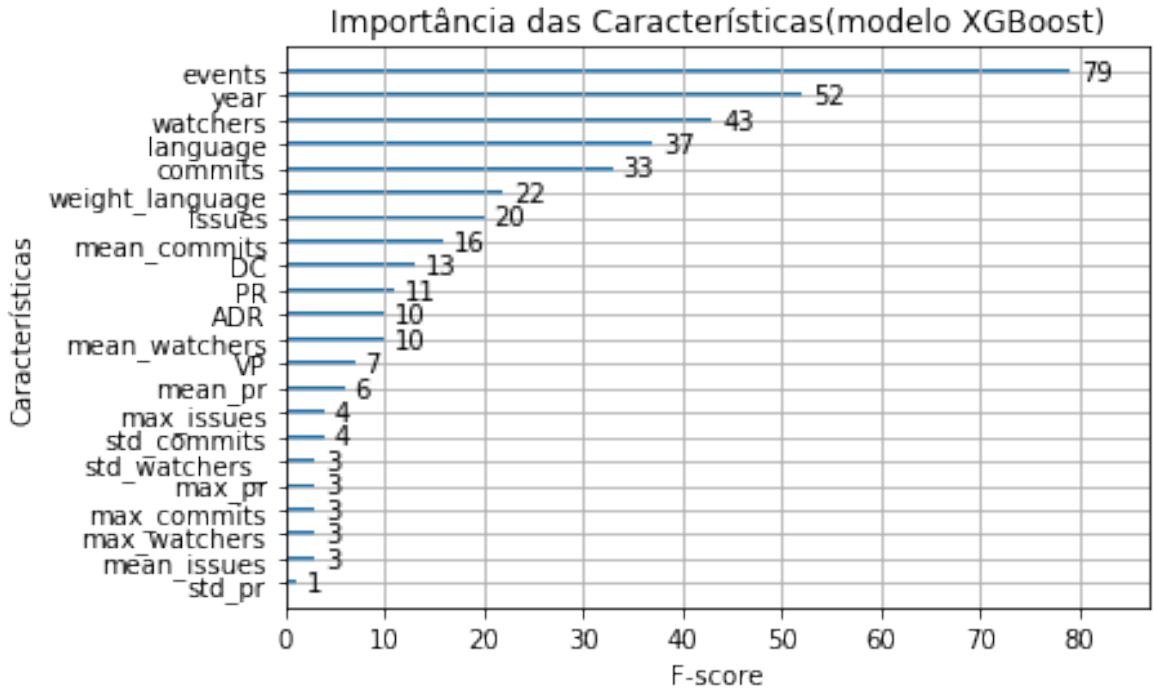


Figura 6.5: Resultado da execução das características mais relevantes para o modelo XGBoost (*weight*)

Além do *weight*, decidimos observar as outras duas medidas, como mostram as Figuras 6.6 e 6.7, para verificar se elas mantêm o mesmo padrão para o *weight*. Como identificado, vemos que a ordenação de importância das características é diferente para a medida de *weight*, quando comparado com *cover* e *gain*. Temos que para *weight*, a principal característica é *events*. Mas, tanto para a medida de *cover* e *gain*, temos que a característica mais relevante é *issues*. Como citado anteriormente nesta Seção, os *top-10* projetos mais recomendados para XGBoost apresentara uma média baixa para *watchers*, *commits* e *pull-requests*, mas para o quantitativo de *issues* a média estava bem mais alta, demonstrando que esta característica parecia ser relevante para os projetos recomendados. O que comprovamos agora utilizando as medidas *cover* e *gain*.

Os métodos de *weight*, *cover* e *gain* são métodos de atribuição de característica vistos de uma forma global, mas é possível analisar as características de forma individualizada. Para verificar a consistência, utilizamos um algoritmo prático e rápido que utiliza a função SHAP (*SHapley Additive exPlanations*) [69], que atribui a cada característica um valor de importância para uma predição específica. Com esta nova abordagem, voltamos à tarefa de interpretar o nosso modelo XGBoost por meio das características exibidas na Figura 6.8.

Após aplicarmos SHAP, verificamos que a característica *issues* é realmente a mais im-

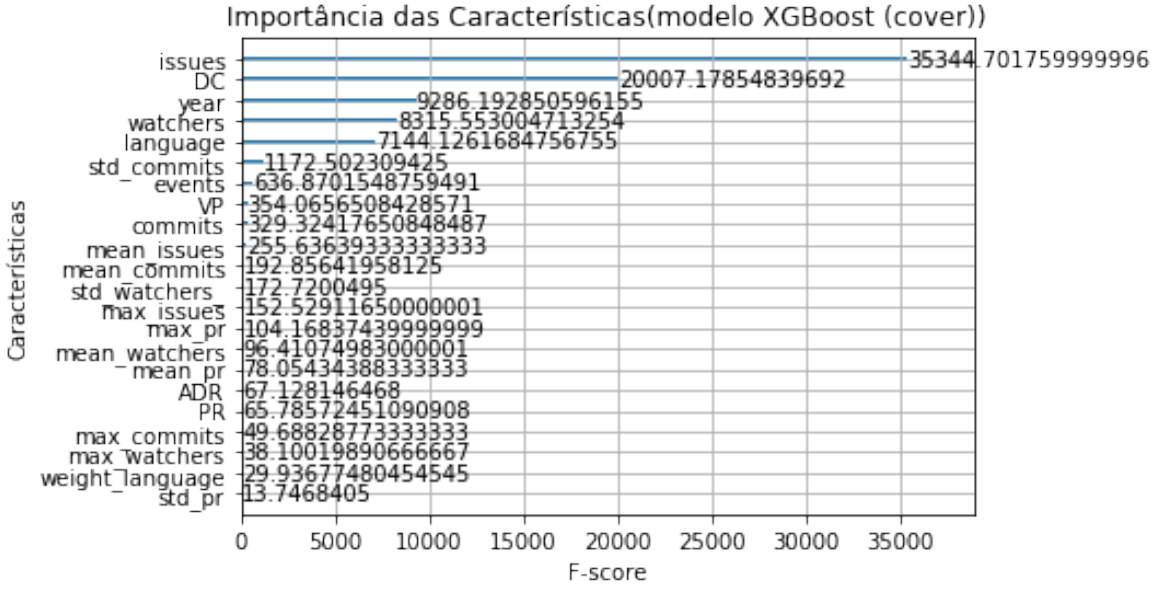


Figura 6.6: Resultado da execução das características mais relevantes para o modelo XGBoost (*cover*)

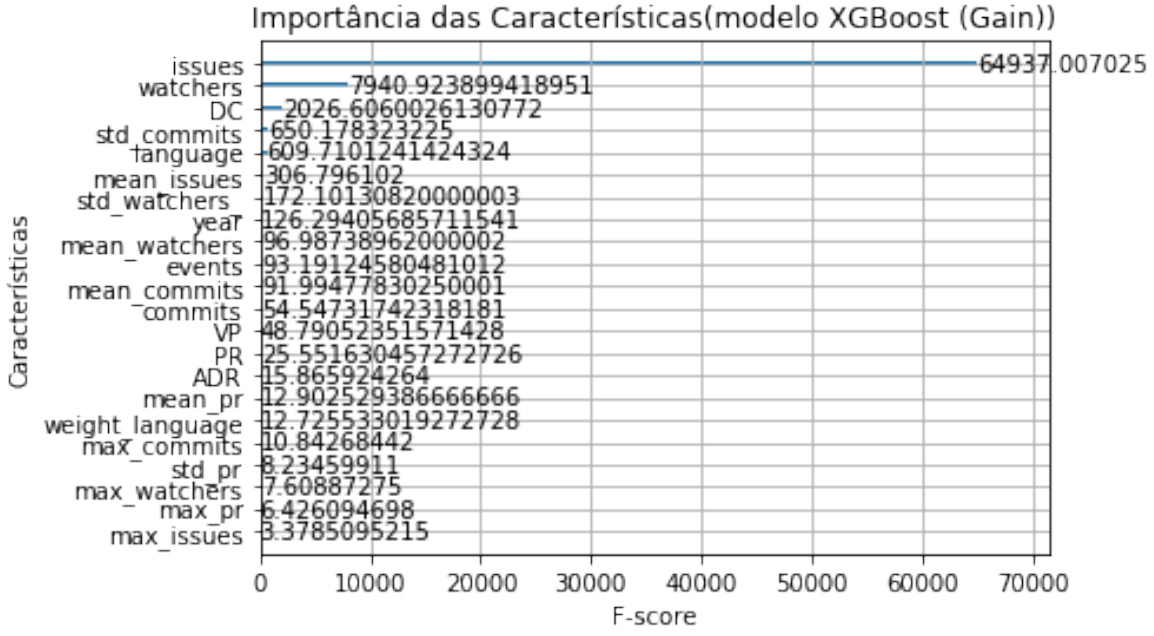


Figura 6.7: Resultado da execução das características mais relevantes para o modelo treinado XGBoost (*gain*)

portante, seguida pela característica de observadores (*watchers*). Como os valores de SHAP têm uma maior consistência, então não nos preocupamos com as contradições entre *events* (*weight*) e *issues* (*cover* e *gain*). No entanto, como agora temos explicações individualizadas para cada projeto, onde podemos fazer mais do que apenas criar um gráfico de barras, traçamos a importância da característica para cada projeto em nosso conjunto de dados, como

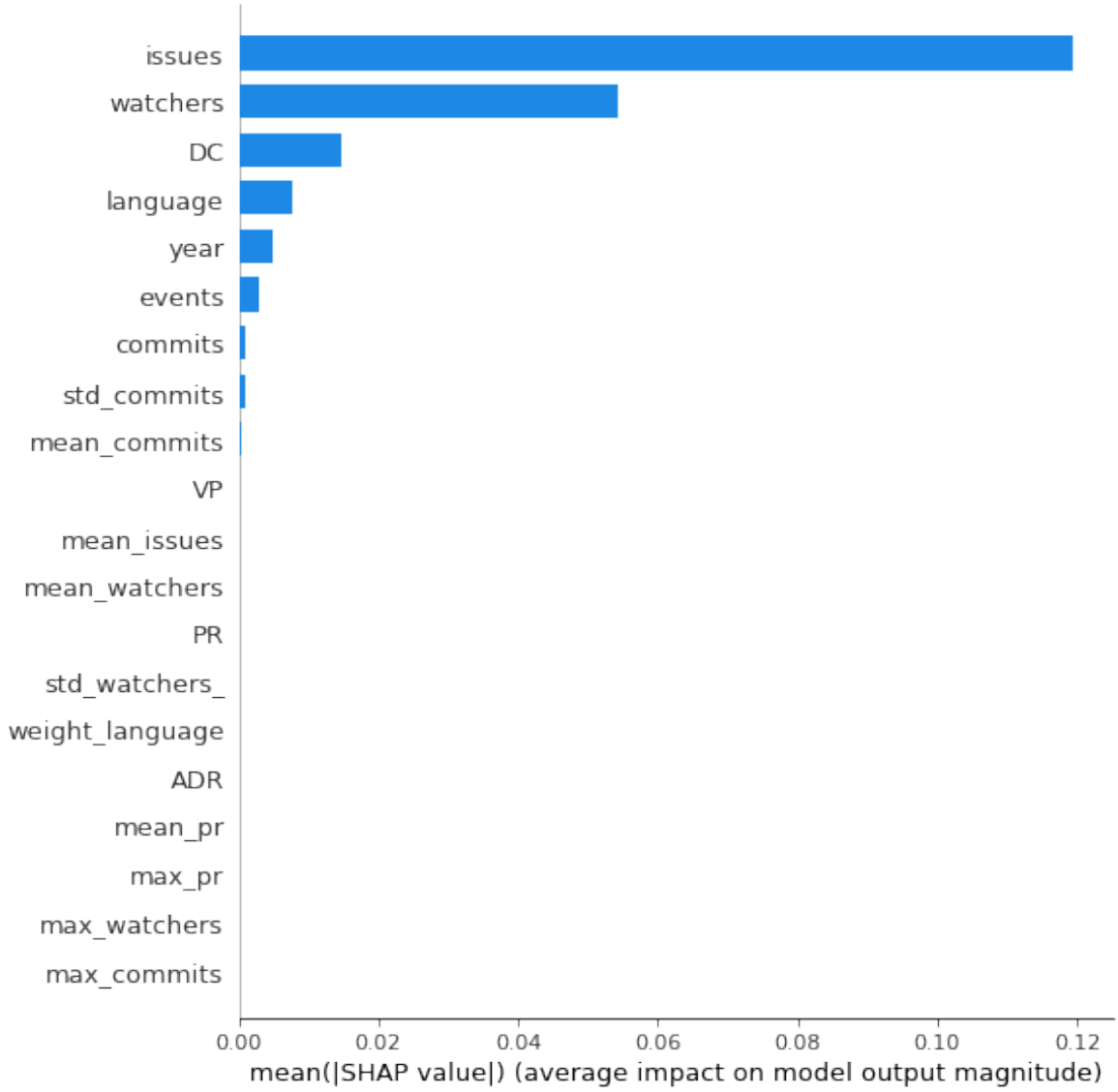


Figura 6.8: Alteração de magnitude média Shap

gostaríamos de definir para responder a questão de pesquisa QP2. Primeiro, utilizamos o modelo *TreeExplainer* disponibilizado pelo pacote SHAP para explicar cada predição e plotamos as características de forma individualizada, como mostra a Figura 6.9.

Dentro do contexto dos projetos mais recomendados usando a técnica XGBoost, os recursos são classificados por média e vemos novamente a característia *issues* como a mais forte na predição de projetos. Ao plotar o impacto de uma característica na amostra, podemos visualizar importantes efeitos discrepantes. Por exemplo, embora a linguagem de programação não seja a característica mais importante globalmente, ela é uma característica muito importante para um subconjunto de projetos. A coloração por valor nas características

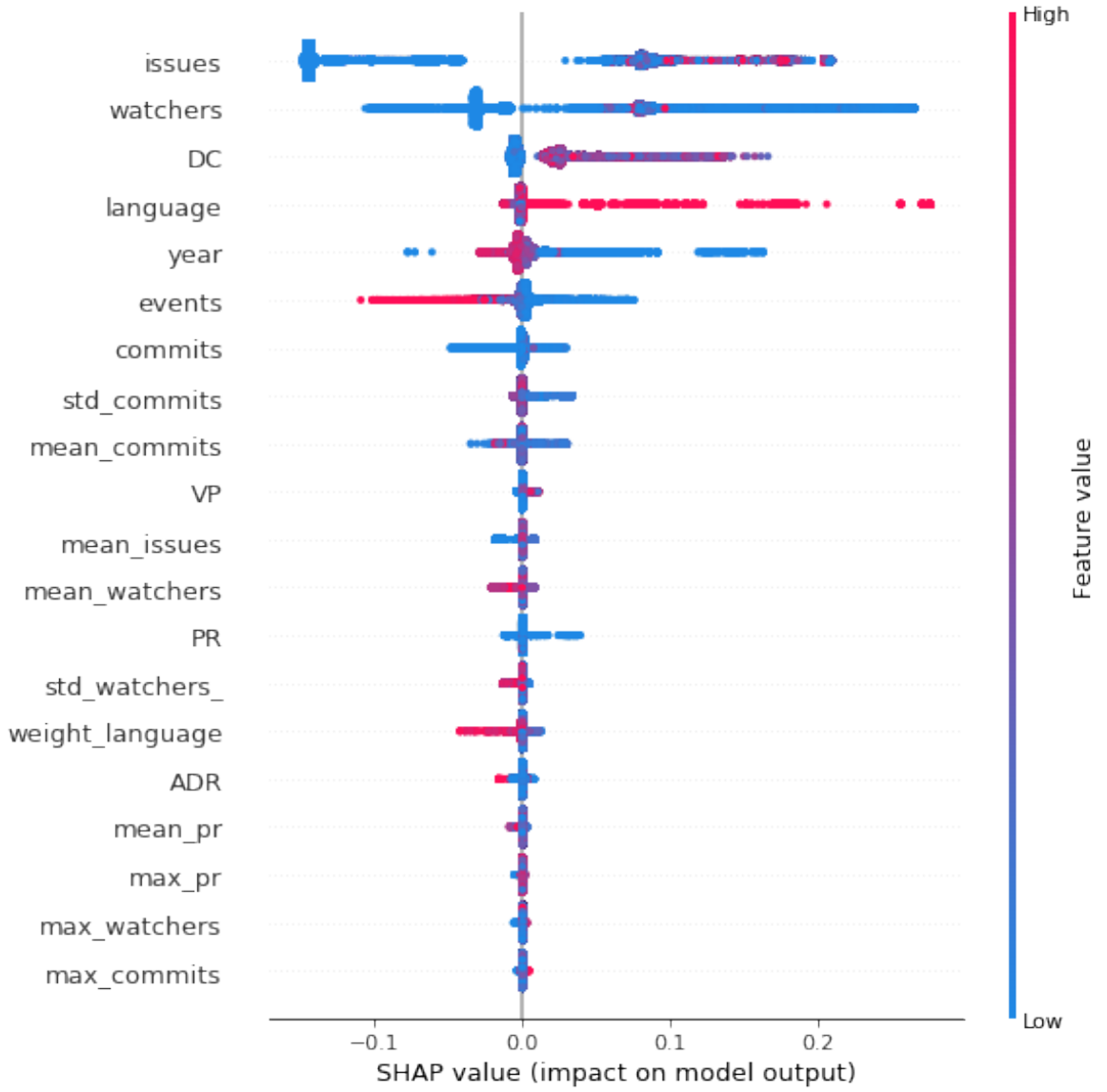


Figura 6.9: Alteração de Magnitude Média (*Tree SHAP*)

nos mostram padrões como o ano, a linguagem de programação, a contribuição diária do projeto e a quantidade de observadores tendo uma chance maior de ser utilizado no aprendizado do modelo, enquanto a quantidade de *pull-requests* não aumenta muito as chances de recomendação de um projeto.

Também analisamos as relações de proximidade entre as características, onde verificamos quanto uma característica influencia a outra no resultado final. Na Figura 6.10, podemos perceber que as características *pull-requests*, *issues*, *watchers* e *commits* são bastante próximas, justificando a relação dessas com *events* (que é uma característica derivada delas). As demais características mostram-se bem próximas em relação ao grupo estatístico. Portanto,

capturando os resultados anteriores, onde *issues*, *watchers* e *year* são características bem relevantes para os resultados, vemos que a correlação delas com as demais, permite que *commits*, *events*, *language* e DC tenham também uma influência maior nos resultados que as demais características.

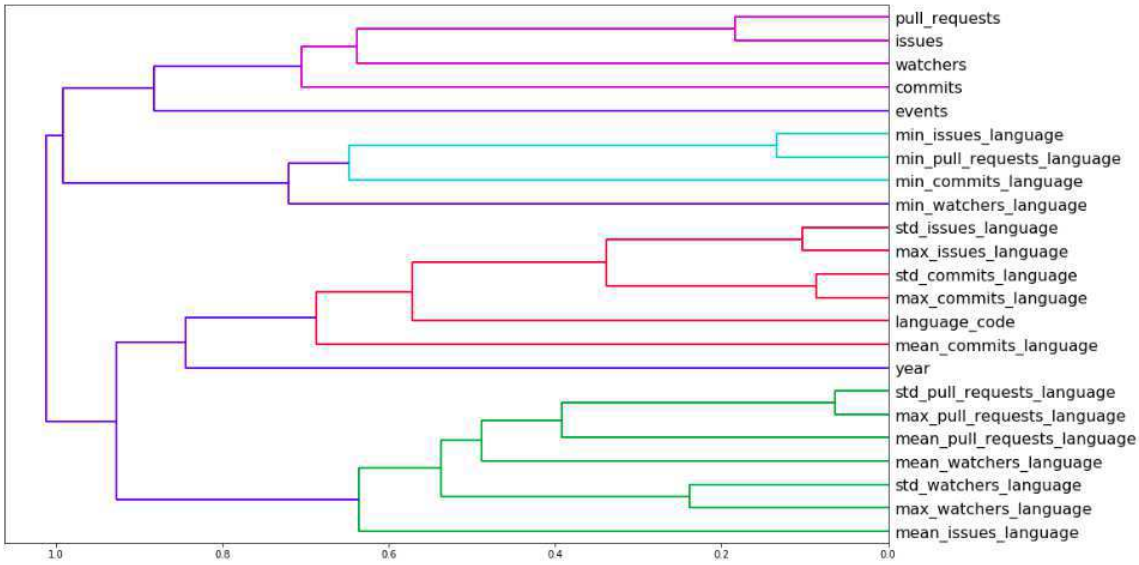


Figura 6.10: Relação de proximidade entre as características

Decidimos executar os modelos de classificação supervisionada (Item-KNN, Random-Forest, XGBoost, MF), utilizando a amostragem negativa e contendo no conjunto de treino apenas as 10 (dez) características mais relevantes listadas para XGBoost, para verificarmos se os resultados apresentam uma melhor precisão, não apenas para o XGBoost, mas para os demais modelos supervisionados.

Por meio do conjunto de teste, validamos a qualidade de recomendação dos modelos. Utilizamos as métricas *Precision*, *Recall* e *F-Measure*. Para o ranqueamento da lista de recomendação, recuperamos $N$ para os valores: $top - 3$, $top - 5$, $top - 10$, $top - 50$ e $top - 100$, como mostra a Tabela 6.5.

A Figura 6.11 compara os resultados das métricas *f-measure* entre as duas amostragens negativas: uma contém todas as 22 (vinte e duas) características (apresentadas nas Seções 5.3 e 5.4) e a outra contém apenas as 10 (dez) características mais relevantes (apresentadas na Seção 6.1). Como podemos verificar, para o modelo XGBoost, o resultado apresentou-se melhor para *precision* e *recall*. Entretanto, para os modelos Item-KNN e MF, os resultados apresentaram uma pequena melhora. Por fim, para o modelo RF, ocorreu uma queda dos

Tabela 6.5: *Precision*(P), *Recall*(R), *F-Measure*(F-M) para Amostragem Negativa

| | Dataset | Amostragem com Dez Características | | | Amostragem com Todas Características | | |
|---|---|---|---|---|---|---|---|
| | | P | R | F-M | P | R | F-M |
| Item-KNN | top-3 | 0.00093 | 0.00028 | 0.00043 | 0.00004 | 0.00003 | 0.00005 |
| | top-5 | 0.00058 | 0.00029 | 0.00038 | 0.00058 | 0.00007 | 0.00012 |
| | top-10 | 0.00034 | 0.00034 | 0.00034 | 0.00087 | 0.00020 | 0.00033 |
| | top-50 | 0.00013 | 0.00066 | 0.00022 | 0.00087 | 0.00102 | 0.00093 |
| | top-100 | 0.00019 | 0.00198 | 0.00036 | 0.00087 | 0.00217 | 0.00130 |
| RF | top-3 | 0.00066 | 0.00004 | 0.00008 | 0.00560 | 0.00004 | 0.00007 |
| | top-5 | 0.00050 | 0.00006 | 0.00010 | 0.00080 | 0.00009 | 0.00017 |
| | top-10 | 0.00058 | 0.00014 | 0.00022 | 0.00099 | 0.00230 | 0.00038 |
| | top-50 | 0.00048 | 0.00059 | 0.00053 | 0.00077 | 0.00092 | 0.00083 |
| | top-100 | 0.00067 | 0.00162 | 0.00095 | 0.00099 | 0.00138 | 0.00081 |
| XGBoost | top-3 | 0.0298 | 0.00284 | 0.00529 | 0.00359 | 0.00283 | 0.00331 |
| | top-5 | 0.0323 | 0.00298 | 0.00552 | 0.01270 | 0.00289 | 0.00391 |
| | top-10 | 0.0389 | 0.00366 | 0.00669 | 0.01270 | 0.00305 | 0.00492 |
| | top-50 | 0.0445 | 0.00431 | 0.00785 | 0.02401 | 0.00428 | 0.00516 |
| | top-100 | 0.0485 | 0.00554 | 0.00994 | 0.03906 | 0.00557 | 0.00631 |
| MF | top-3 | 0.00011 | 0.00010 | 0.00010 | 0.00010 | 0.00001 | 0.00001 |
| | top-5 | 0.00010 | 0.00005 | 0.00006 | 0.00010 | 0.00001 | 0.00002 |
| | top-10 | 0.00006 | 0.00006 | 0.00006 | 0.00011 | 0.00002 | 0.00004 |
| | top-50 | 0.00012 | 0.00011 | 0.00012 | 0.00010 | 0.00015 | 0.00012 |
| | top-100 | 0.00005 | 0.000025 | 0.00025 | 0.00011 | 0.00030 | 0.00015 |

valores. Onde podemos constatar que as 10 (dez) características mais relevantes apresentam uma melhoria relevante nos resultados para o modelo XGBoost. Por exemplo, agora para *top-10* o valor de *precision* é 0.0389, e anteriormente o valor era de 0.01270. Observando, também, o valor de *f-measure* para *top-100*, temos que o valor anterior era de 0.00631 e agora passou a ser 0.00994.

Decidimos analisar as posições de acerto das predições para os quatro modelos propos-

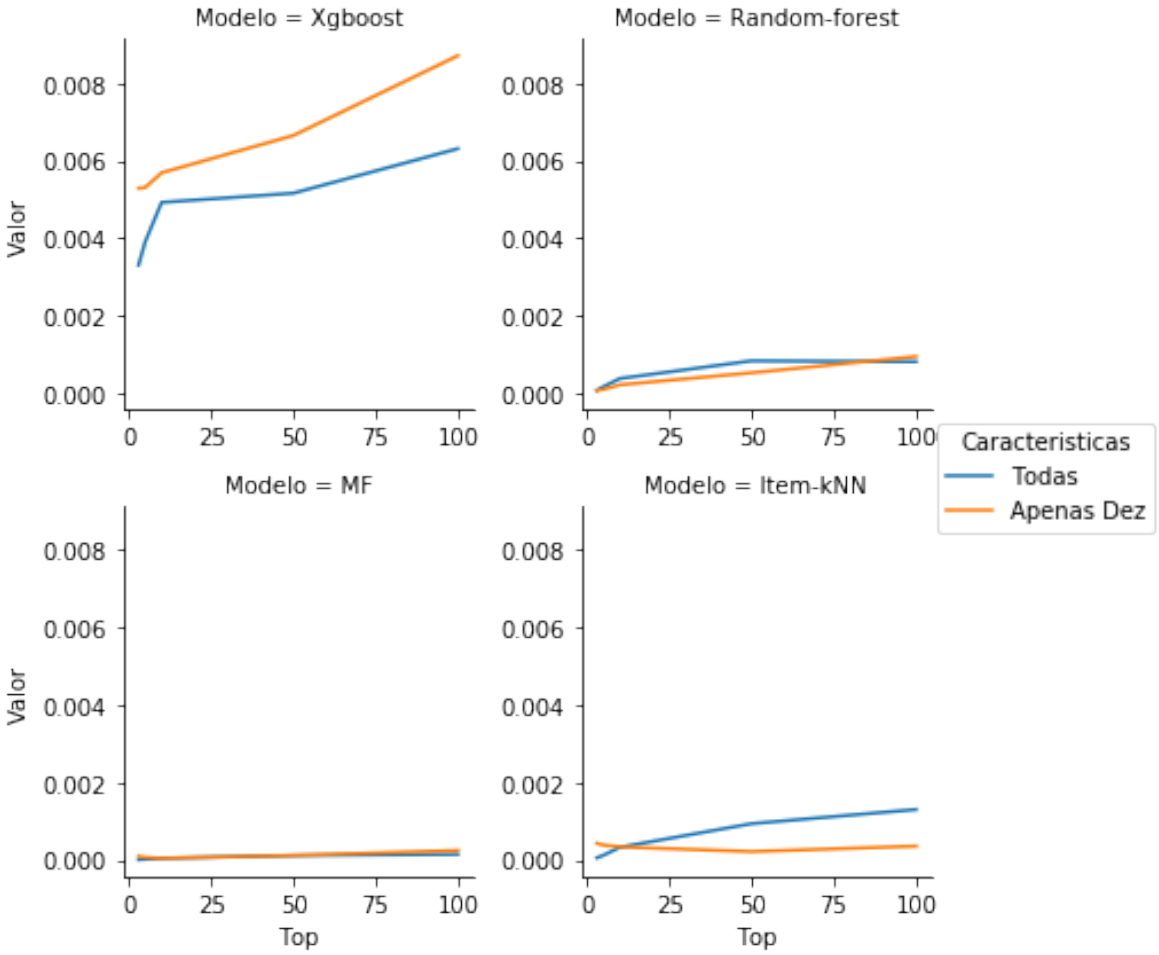


Figura 6.11: Comparação entre Modelos usando F-Measure para 10 (dez) Características

tos para verificarmos quais deles têm resultados mais próximos das posições $top - 3$ até $top - 100$. Como podemos ver na Figura 6.12, o modelo XGBoost acerta muito mais que os demais. O modelo RF não apresenta o mesmo desempenho que o modelo XGBoost, mas demonstra acertar mais nas predições concentradas na posição no top-3 até top-60. O pior caso apresentado é o modelo MF que apresenta suas recomendações mais concentradas acima de top-100.

Um outro ponto a ser analisado são os acertos para as predições. No modelo XGBoost, vemos que o modelo apresenta cerca de 12000 predições entre o *top-1* e *top-5*. Vemos que o RF tem uma faixa de acerto até o *top-100*, mas, no geral, acerta apenas um pouco mais de 16 vezes entre *top-10* e *top-20*. Os modelos Item-KNN e MF não diferem muito do modelo RF.

Para resposta da nossa questão de pesquisa QP2, conseguimos identificar as características mais relevantes, onde estas sendo reaplicadas aos modelos, melhoraram os resultados

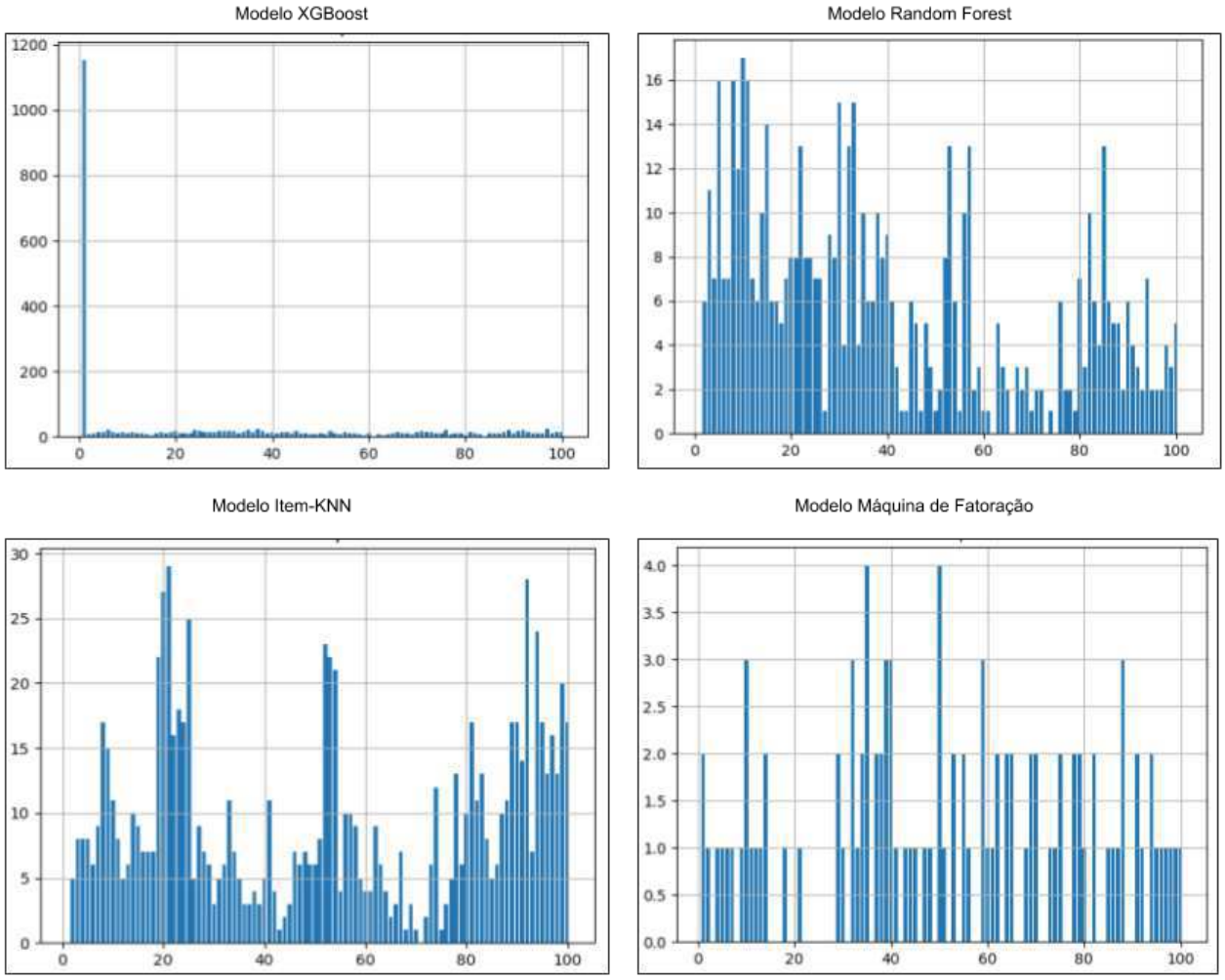


Figura 6.12: Predições para o Modelos

para o XGBoost.

## 6.3 Nova Proposta de Aprendizado Para Classificação

Para respondermos QP1 e QP2, inicialmente, investigamos o aprendizado dos modelos a partir das 22 características disponibilizadas nas duas amostragens de dados selecionadas. Analisamos também a importância das características no aprendizado dos modelos e quais apresentaram uma maior relevância para a qualidade das recomendações de projetos. Verificamos que todas as características foram relevantes, mas observamos que dez delas tiveram um maior destaque. Entretanto, os resultados ainda não se mostravam promissores e, por este motivo, decidimos experimentar uma nova proposta de definição do conjunto de teste a partir

da amostragem negativa. Para isso, definimos uma nova questão de pesquisa e re-executamos os experimentos.

- QP3 - Qual o efeito nos fatores de qualidade das recomendações, quando aplicada uma nova metodologia de teste a partir da amostragem negativa?

Para respondermos QP3, decidimos avaliar uma nova metodologia de teste e predição dos dados adotada por Cremonesi et al. [29]. A proposta desta metodologia é utilizar uma classificação, onde para cada projeto conhecido pelo usuário, selecionamos um conjunto de projetos desconhecidos e aplicamos um modelo diferenciado para o conjunto de teste. Nosso objetivo, para responder QP3, é encontrar uma solução alternativa para o aprendizado da classificação e verificar se os resultados da recomendação poderiam ser melhores se definirmos o conjunto de dados de uma forma diferente. Neste caso, estamos transformando o problema da recomendação em um problema de classificação por meio da construção de um modelo de pontuação diferente.

A proposta é a criação de uma nova amostragem para realização de uma nova forma de classificação. Para Cremonesi et al. [29], uma construção aleatória de um grande conjunto de dados tornaria a predição fortemente influenciada para os itens mais populares. A solução seria excluir os itens mais populares e conhecidos pelos usuários do conjunto de dados e manter os projetos conhecidos de cada usuário. A partir de cada projeto, é criada uma amostra e realizada a predição deste subconjunto de dados. Assim, a amostra resultante enfatizaria os itens não triviais e que também se mostram importantes para o usuário, moldando-se às novas expectativas e analisando projeto por projeto.

Desta forma, geramos um novo conjunto de dados de treinamento $M$ e um novo conjunto de dados de teste $T$. O conjunto de teste $T$ contém apenas os projetos do usuário contendo os projetos que são *fork* (*target* com valor 1). Na predição, para cada item $i$ no conjunto de $T$ para o usuário $u$ realizamos os seguintes passos:

1. Selecionamos aleatoriamente 1000 (mil) projetos adicionais não classificados pelo usuário $u$ disponibilizados pela plataforma GitHub. Ou seja, o usuário não realizou *fork* destes projetos. Assim, assumimos que o usuário $u$ não teria interesse ou desconhecia esses 1000 projetos.

2. A partir do aprendizado no conjunto de treino $M$, predizemos os 1000 projetos + o item $i$ de projeto do usuário $u$, que tem *fork* igual a 1. Ou seja, predizemos 1001 projetos.

3. Com isso, geramos uma lista classificada e ordenamos de acordo com o valor da predição. Verificamos a posição $p$ do item $i$ na lista ordenada. O melhor resultado seria o item $i$ preceder todos os demais itens da lista, ou seja, a posição do item $i$ ser o primeiro lugar ($p = 1$). Desta forma, saberíamos que os modelos propostos estariam aprendendo corretamente e de acordo com as preferências do usuário.

4. Para cada top-N da lista classificada: se $p <= N$, então temos que o item $i$ está na lista de projetos a serem recomendados. Caso contrário, não teremos o item $i$ recomendado.

Aplicamos a nova metodologia de teste para investigar os quatro modelos de aprendizado de máquina e realizarmos a recomendação. Por meio do conjunto de teste, validamos os fatores de qualidade dos modelos para identificarmos quais deles teve uma maior precisão. Utilizamos as métricas *Precision*, *Recall* e *F-Measure*, como mostra a Tabela 6.6. Para o ranqueamento da lista de recomendação, recuperamos $N$ para os valores: $top - 3$, $top - 5$, $top - 10$, $top - 50$ e $top - 100$.

Observando os resultados da Tabela 6.6, para a métrica *F-M*, os modelos XGBoost, RF e MF alcançam para $top - 3$ bons resultados para recomendação dos projetos. Vemos que *precision* para os modelos MF e RF têm resultados iguais a 0,33 e 0,30, respectivamente. Isso significa que cerca de 33% do total dos projetos recomendados são apresentados entre as três principais recomendações. Um resultado revelador foi para *MF*, que nesta nova metodologia apresentou resultados bem superiores aos anteriormente analisados nas Seções 6.1 e 6.2. Tivemos uma diferença de *precision* de 0,00011 (de resultados anteriores) para 0,3. Ou seja, *MF* incorporou as informações das características com maior qualidade, quando reduzimos a dimensionalidade destas. Ao contrário, o modelo *Item-KNN* apresentou resultados bem inferiores aos anteriormente analisados. Portanto, a proposta de uma nova estrutura para o aprendizado da classificação mostrou-se promissora e tivemos resultados mais representativos para a recomendação de projetos. Apresentamos uma comparação entre *F-Measure* para as três questões de pesquisa no Apêndice A.

Tabela 6.6: *Precision*(P), *Recall*(R), *F-Measure*(F-M)

| Dataset | | Novo Aprendizado da Classificação | | |
|---|---|---|---|---|
| | | P | R | F-M |
| Item-KNN | top-3 | 0.000006 | 0.000002 | 0.000009 |
| | top-5 | 00.000006 | 0.000002 | 0.000006 |
| | top-10 | 0.000004 | 0.000002 | 0.000003 |
| | top-50 | 0.000002 | 0.000002 | 0.0000007 |
| | top-100 | 0.000004 | 0.000002 | 0.0000003 |
| RF | top-3 | 0.3050 | 1 | 0.4674 |
| | top-5 | 0.1796 | 1 | 0.3045 |
| | top-10 | 0.0885 | 1 | 0.1626 |
| | top-50 | 0.0174 | 1 | 0.0343 |
| | top-100 | 0.0087 | 1 | 0.0172 |
| XGBoost | top-3 | 0.4624 | 1 | 0.4620 |
| | top-5 | 0.1762 | 1 | 0.2997 |
| | top-10 | 0.0866 | 1 | 0.1594 |
| | top-50 | 0.0170 | 1 | 0.0335 |
| | top-100 | 0.0085 | 1 | 0.0168 |
| MF | top-3 | 0.3333 | 1 | 0.4999 |
| | top-5 | 0.1999 | 1 | 0.3333 |
| | top-10 | 0.0999 | 1 | 0.1818 |
| | top-50 | 0.0199 | 1 | 0.0392 |
| | top-100 | 0.0099 | 1 | 0.0197 |

Por fim, voltamos a analisar as posições das recomendações para os quatro modelos propostos até a posição top-100. Como podemos ver na Figura 6.14, o modelo XGBoost melhorou as recomendações para a posição *top-3*. O modelo RF saiu de uma média de 16 recomendações para mais de 12000 em *top-3*. O mesmo ocorreu para o modelo MF, que melhorou mais nas recomendações para *top-3*. Entretanto, o modelo Item-KNN piorou o desempenho em relação aos resultados apresentados anteriormente.

A nova proposta de geração do conjunto de teste contendo apenas projetos conhecidos

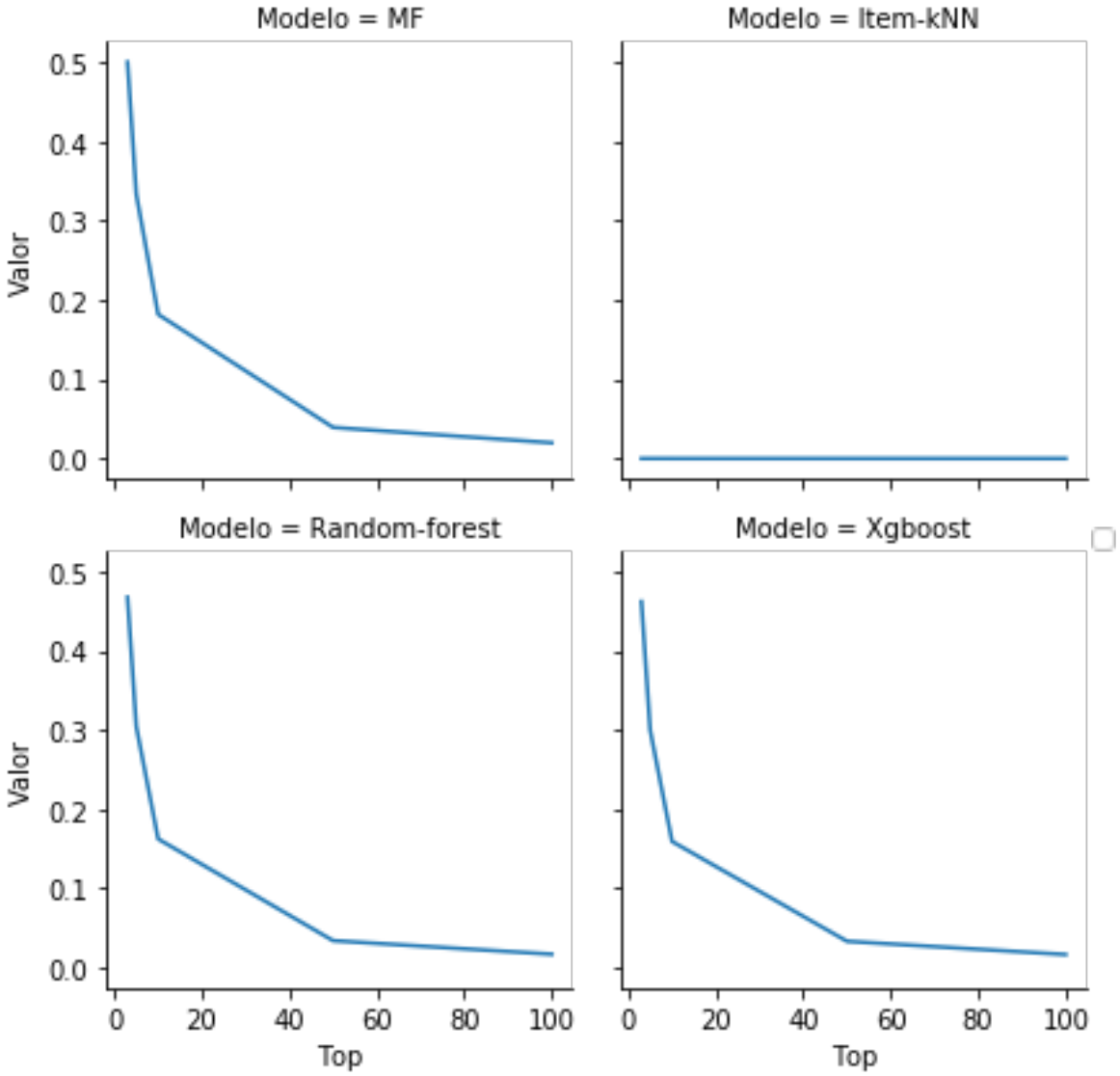


Figura 6.13: Resultados da Métricas para Aprendizado da Classificação

pelos usuários e uma busca aleatório de projetos desconhecidos pelo usuário no conjunto de treinamento, permitiu um melhor aprendizado dos modelos para as 22 características propostas neste estudo. Para responder a questão de pesquisa QP3, os fatores de qualidade das recomendações apresentaram resultados relevantes e melhores quando propomos uma nova abordagem de preparação do conjunto de dados e que impactou diretamente o aprendizado da classificação.

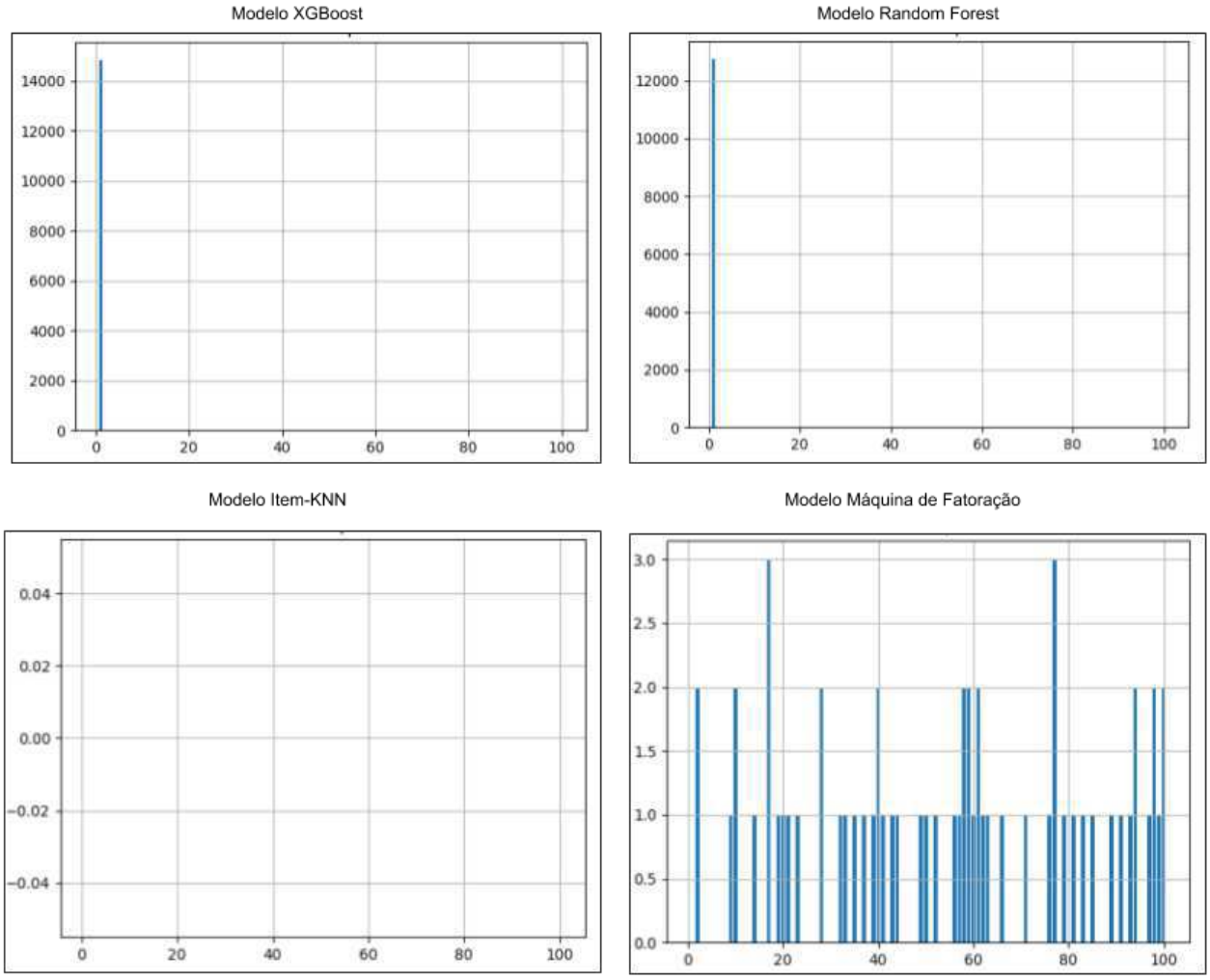


Figura 6.14: Posições das Recomendações para o Modelos na Nova Proposta para o Aprendizado da Classificação

## 6.4 Ameaças à Validade

Uma ameaça de validade interna comum é a parametrização aplicada ao estudo. Testamos algumas possibilidades de parametrização, mas estas não foram exaustivas. Portanto, preferimos deixar os valores padrões definidos para cada parâmetro nos algoritmos utilizados [95]. Provavelmente, para mitigar esta ameaça, seria necessário mais estudos para definirmos a melhor parametrização aplicada a cada algoritmo, adequado para o contexto de recomendação de projeto no GitHub.

Acreditamos que os algoritmos utilizados foram adequados para o nosso experimento. Certamente, existem diferentes implementações para esses algoritmos e várias outras opções possíveis, mas eles são viáveis de serem implementados e representativos para o contexto

dos sistemas de recomendação. Os algoritmos escolhidos também são bem explorados em outros contextos de pesquisa além do nosso, e achamos que eles seriam bem aplicados no contexto deste trabalho. O conjunto de características selecionadas também é algo a ser considerado como uma possível ameaça à validade de construção, pois não exaurimos todas as possibilidades e características como, *star* e navegação do usuário na plataforma não foram aplicadas neste estudo devido a limitação de dados na base de dados do GHTorrent. Acreditamos que atenuamos essa ameaça selecionando um conjunto de características representativas de serem identificadas para usuários e projetos.

No escopo deste trabalho não desenvolvemos uma aplicação do sistema de recomendação de forma on-line. Desta forma, não conseguimos validar pessoalmente com cada usuário se o recomendador estava recomendando de acordo com o intersse deste. Também tivemos uma quantidade extensa de dados e muitos projetos apresentaram-se como *outliers*, por isso foi preciso minimizar esse problema tratando os dados de usuários e projetos, sendo necessária a exclusão dos *outliers*. O tratamento destes dados levaram mais tempo do que o planejado. Em experimentos futuros, planejamos aplicar a recomendação on-line e qualificar com uma qualidade ainda melhor o conjunto de dados de usuários e projetos.

## 6.5 Considerações Finais

Neste capítulo apresentamos os resultados dos experimentos e demonstramos a eficácia dos métodos propostos para a recomendações de projetos na plataforma GitHub. Em relação à questão de pesquisa (QP1), as métricas propostas apresentaram resultados promissores para XGBoost e RF, e não tão promissores para FBC, Item-KNN e MF. Por causa do comportamento inicial dos modelos propostos, decidimos analisar o comportamento do aprendizado destes modelos perante as 22 características propostas, demonstrando que estas adicionam poder preditivo e constribui para que um sistema de recomendação forneça melhores recomendações para os usuários da plataforma, mas, ainda assim, não foi um resultado tão diferenciado como esperávamos, como mostramos na questão de pesquisa (QP2). Decidimos por ajustar os conjuntos de dados, primeiramente, realizando o aprendizado de máquina apenas com as 10 (dez) características mais relevantes. Com este novo conjunto de dados melhoramos as recomendações analisadas, mas os resultados não demonstraram uma dife-

rença representativa. Por fim, para responder nossa questão de pesquisa (QP3), propomos uma nova metodologia de teste para o aprendizado da classificação. Obtivemos resultados diferenciados, permitindo uma melhoria significativa do recomendador, onde tivemos recomendação de prejetos entre o $top - 10$ da lista.

# Capítulo 7

# Conclusões

O objeto de estudo da pesquisa reportada neste documento foi a recomendação de projetos na plataforma GitHub. Propomos a avaliação da qualidade das recomendações de projetos geradas a partir das predições realizadas pelas técnicas de classificação aplicadas às diversas características de usuários e projetos extraídas dos dados e metadados disponíveis pela plataforma GitHub. Para melhor compreensão de cada tipo de característica e enquadramento das informações, propomos o agrupamento das características em *atratividade*, *visibilidade*, *medidas estatísticas* e *engajamento*. De modo geral, o objetivo do trabalho foi experimentar um conjunto de algoritmos de diferentes paradigmas aplicados aos perfis de usuários e projetos, que são combinados, para realizar a predição. Após isso, foi verificado os fatores de qualidade dos modelos, que determina o sucesso do recomendador.

Inicialmente, foi realizada uma pesquisa sobre sistemas de recomendação, relatada no Capítulo 2, por meio da discussão dos conceitos de seus principais elementos, abordagens, modelos e estratégias para entendermos como os recomendadores podem ser utilizados pela rede social GitHub. Enfatizamos uma ampla literatura de conceitos e teorias desenvolvidas nessas áreas e que servem de base teórica para o estudo conduzido neste trabalho.

Apresentamos o ecossistema GitHub por meio de características sociais, colaboração e eventos. A organização desse ecossistema e as análises dos artigos apresentados neste capítulo desempenharam diversos papéis em pesquisas científicas e desenvolvimento de sistemas recomendação ([92]), como, por exemplo: (i) servir como um guia de aspectos que devem ser considerados por pesquisadores e desenvolvedores, (ii) prover métricas de recomendação por meio dos quais os pesquisadores podem realizar novas descobertas e raciocinar sobre a

área em estudo, e (iii) justificar e motivar o desenvolvimento de novas estratégias de modelos de recomendação. Todos os estudos analisados se concentram na recomendação de projetos direcionada para certas características, seja do usuário ou projeto. Com base nesta pesquisa, propomos uma análise de dados centrada na diversidade das características da plataforma GitHub. Em nossa proposta, apresentamos as características que contêm informações sobre os dados contextuais, como é o caso do conjunto de atratividade, que são os fatores relacionados à popularidade dos usuários e projetos; de visibilidade, que é disponibilizada por uma plataforma que promove a transparência da informação; do conjunto de engajamento, que foi definido a partir de métricas derivadas das informações de atividade dos projetos; e do conjunto de estatísticas, que exibe a distribuição dos dados.

Por meio da exploração dos dados, identificamos a necessidade de recomendação de projetos baseados na característica alvo chamada *fork*. A partir dessa informação, conseguimos identificar a possibilidade de crescimento na colaboração de projetos entre usuários. Com isso, demonstramos a importância dos repositórios, seu crescimento e atualizações dentro da plataforma GitHub. Apresentamos algumas dificuldades dos usuários em encontrar projetos que sejam do seu interesse e como os sistemas de recomendação podem ajudar neste processo de identificação de projetos de interesse do usuário. Também definimos algumas características principais e os diferentes conjuntos de dados gerados para atendermos ao objetivo principal deste trabalho.

Discutimos o uso da classificação não supervisionada e supervisionada para os conjuntos de dados definidos para usuários e projetos da plataforma GitHub. Contextualizamos os conceitos das características sociais e criamos outras características derivadas a partir da análise da linha do tempo dos projetos quanto ao engajamento das contribuições aplicadas nos projetos. Demonstramos os modelos a serem utilizados para o aprendizado da classificação e, por fim, apresentamos as métricas a serem utilizadas para validação dos resultados.

Os resultados dos experimentos demonstraram a eficácia da abordagem proposta para a recomendações de projetos na plataforma GitHub. As métricas propostas apresentaram resultados promissores para alguns modelos (XGBoost e RF) e não tão promissores para outros (Item-KNN, FBC e MF) (QP1). Também analisamos o comportamento do aprendizado dos modelos perante as características propostas, demonstrando que estas adicionam poder preditivo e constribui para que um sistema de recomendação forneça melhores recomenda-

ções para os usuários da plataforma, mas, ainda assim, não foi apresentado um resultado tão diferente do obtido anteriormente (QP2). Propomos um novo conjunto de dados para melhorarmos as recomendações analisadas e uma nova abordagem foi aplicada para o aprendizado da classificação. Desta forma, obtivemos resultados diferenciados, permitindo a recomendação entre o $top - 10$ da lista de contribuição (QP3).

Por fim, avaliamos nossa proposta no contexto de recomendações de projetos na plataforma GitHub. A análise revelou que a abordagem no uso das características sociais, como também o uso de características derivadas a partir do comportamento dos projetos, contribui para qualidade das recomendações. Após a análise dos dados da plataforma, implementação das características, execução dos modelos e análise dos experimentos com dados reais, foi possível concluir que a utilização de atributos sociais em modelos supervisionados do estado-da-arte podem superar a qualidade das recomendações para modelos não supervisionados. Além disso, a nova proposta de aprendizado da classificação supera efetivamente a qualidade das recomendações nos aspectos de *precision*, *recall* e *f-measure*.

## 7.1 Contribuições

Em resumo, as principais contribuições deste trabalho de pesquisa foram:

1. A investigação das questões de pesquisa declaradas relacionadas à eficácia dos algoritmos de recomendação baseados contribuição de projetos na plataforma GitHub;

2. A análise contextual abrangente da rede social GitHub e seu amplo conjunto de características, extraídas de dados de usuários e repositórios, para representar as preferências do usuário por projetos para recomendações baseadas em aprendizado de máquina;

3. A adaptação de métricas de atividade dos projetos no GitHub: atividade do projeto, contribuição diária em um projeto e variação da periodicidade [23]. Tais métricas se ajustaram ao objetivo de analisar a periodicidade de atualização do projeto e a duração das contribuições dos colaboradores nos projetos, como também apresentaram-se relevantes no aprendizado dos modelos, principalmente a métrica de contribuição diária;

4. A avaliação dos recomendadores propostos com dois conjuntos de dados com amostragem balanceada e negativa, demonstrando que os modelos XGBoost e Random-Forest apresentam um melhor desempenho em relação aos demais modelos do estado-da-arte.

## 7.2 Limitações e Trabalhos Futuros

A pesquisa reportada neste documento não é exaustiva em estudar a plataforma GitHub e todas as suas características, nem em extrair métricas quantitativas a partir dessas características. Também não foi possível analisar os projetos mais representativos da plataforma GitHub em diferentes contextos de uso e atualizações. Não realizamos um estudo detalhado da parametrização dos algoritmos propostos e desempenho computacional dos algoritmos, o que limita o nosso poder de conclusão dos resultados. Realizamos toda a avaliação em torno de *precision*, *recall* e *f-measure*.

Novas pesquisas podem ser conduzidas nessa direção e, para trabalhos futuros, podemos considerar a análise de outros recursos que se mostrem úteis na análise de outras características fornecidas pela plataforma para identificação das preferências dos usuários. A extração de novas métricas também pode ser considerada, não apenas focando nas atualizações dos projetos, mas também na permanência dos contribuidores na plataforma ou na navegação dos usuários pela plataforma. Ou seja, é importante colocar em perspectiva que novos tipos de conhecimento sobre as informações disponibilizadas podem ser obtidos ao utilizar outras características da plataforma GitHub. A complementação de novas características analisadas neste estudo, só tem a agregar no resultado para trabalhos futuros.

Considerando os resultados reportados neste documento e as pesquisas que têm sido realizadas por trabalhos relacionados, mostra-se relevante a condução de mais estudos qualitativos que investiguem relações entre os os usuários e projetos. Esse tipo de estudo pode direcionar o desenvolvimento de uma nova geração de estratégias que visem melhorar a recomendação de projetos na plataforma. Tais estratégias podem detectar o comportamento do usuário e incorporar as métricas na dinâmica de navegação do usuário na plataforma de modo a colocar em prática ações que determinem ajudem o usuário a encontrar os projetos de seu interesse. Pois, ao mostrarmos que os usuários se comportam de forma diferente, este estudo motiva o desenvolvimento de estratégias personalizadas na recomendação de proje-

tos. Tais estratégias podem monitorar o comportamento de cada usuário e seus respectivos projetos, e quando necessário, ativar automaticamente uma recomendação de projeto mais adequada. As estratégias podem se concentrar em promover o aumento da colaboração de projetos ou até mesmo estratégias de relações sociais entre os usuários.

Naturalmente as análises conduzidas neste trabalho podem ser ampliadas para contextos ainda mais complexos que tendem a ganhar importância com o uso de aprendizado de máquina cada vez mais difundido. Alguns cenários com tendência de crescimento são: (i) analisar o conteúdo de codificação dos repositórios para melhorias adicionais; (ii) explorar outras redes sociais que utilizam a codificação social e que possuam características semelhantes a este trabalho ou novas características relevantes para a aprendizagem de máquina; (iii) criar sistemas de recomendação híbridos, personalizados e subjetivos (relação entre projetos e impacto do comportamento humano em um ambiente colaborativo); (iv) aplicar as características coletadas em algoritmos de redes neurais para verificarmos a precisão das recomendações; (v) realizar um estudo detalhado para os possíveis resultados apresentados para diferentes parametrizações dos algoritmos aplicados; (vi) realizar análise de desempenho computacional aos algoritmos aplicados; e (vii) comparar o estudo com outros *baselines* apresentados em outros estudos científicos.

Por fim, o estudo de recomendação de projetos na plataforma GitHub, reportado neste documento, mostrou-se relevante para entendermos com maior clareza o comportamento do usuário de uma rede social colaborativa. Sobre nossas conclusões, diversos outros estudos podem ser conduzidos. Esperamos que este trabalho inspire novas pesquisas de recomendação em ambientes colaborativos.

# Bibliografia


[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*, chapter 7, pages 217–253. Springer US, New York, NY, 2011. doi: 10.1007/978-0-387-85820-3.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[4] Santa Agreste, Pasquale De Meo, Emilio Ferrara, Sebastiano Piccolo, and Alessandro Provetti. Analysis of a heterogeneous social network of humans and cultural objects. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(4):559–570, 2015.

[5] Ghadah Alamer and Sultan Alyahya. Open source software hosting platforms: A collaborative perspective's review. *JSW*, 12(4):274–291, 2017.

[6] Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, and Josep M Pujol. Data mining methods for recommender systems. In *Recommender systems handbook*, pages 39–71. Springer, 2011.

[7] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and JosepM. Pujol. Data mining methods for recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 39–71. Springer US, 2011. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_2.

[8] John A Barnes and Frank Harary. Graph theory in network analysis. *Social networks*, 5(2):235–244, 1983.

[9] Christian Bauer and Gavin King. Hibernate in action. 2005.

[10] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

[11] Basiliyos Tilahun Betru, Charles Awono Onana, and Bernabe Batchakui. Deep learning methods on recommender system: A survey of state-of-the-art. *International Journal of Computer Applications*, 162(10):17–22, 2017.

[12] Tegawendé F Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillère. Orion: a software project search engine with integrated diverse software artifacts. In *2013 18th international conference on engineering of complex computer systems*, pages 242–245. IEEE, 2013.

[13] Kelly Blincoe, Francis Harrison, and Daniela Damian. Ecosystems in github and a method for ecosystem identification using reference coupling. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 202–211. IEEE, 2015.

[14] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[15] Hudson Silva Borges. Characterizing and predicting the popularity of github projects. 2018.

[16] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *arXiv preprint arXiv:1301.7363*, 2013.

[17] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[18] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. How do centralized and distributed version control systems impact software changes? In *Procee-*

*dings of the 36th International Conference on Software Engineering*, pages 322–333. ACM, 2014.

[19] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[20] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[21] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: the role of prior social links and language experience. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 817–828. ACM, 2015.

[22] Thaciana GO Cerqueira, Franklin Ramalho, and Leandro Balby Marinho. A content-based approach for recommending uml sequence diagrams. In *SEKE*, pages 644–649, 2016.

[23] Thaciana GO Cerqueira, Leandro Balby Marinho, and Franklin Ramalho. Feature evaluation for project preferences representation in github. In *Proceedings of 15th International Conference on Machine Learning and Data Mining*, volume II, pages 655–668. Springer, 2019.

[24] Nitesh V Chawla. C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML*, volume 3, page 66, 2003.

[25] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[26] Xinxin Chen, Yu Guo, Yang Yang, and Zhenqiang Mi. Trust-based collaborative filtering algorithm in social network. In *Computer, Information and Telecommunication Systems (CITS), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[27] Pham Minh Chuan, Cu Nguyen Giap, Chintan Bhatt, Tran Dinh Khang, et al. Enhance link prediction in online social networks using similarity metrics, sampling, and classification. In *Information Systems Design and Intelligent Applications*, pages 823–833. Springer, 2018.

[28] Paolo Cremonesi, Roberto Turrin, Eugenio Lentini, and Matteo Matteucci. An evaluation methodology for collaborative recommender systems. In *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*, pages 224–231. IEEE, 2008.

[29] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[30] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[31] Adele Cutler, D Richard Cutler, and John R Stevens. Random forests. In *Ensemble machine learning*, pages 157–175. Springer, 2012.

[32] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[33] Antonina Dattolo, Felice Ferrara, and Carlo Tasso. The role of tags for recommendation: a survey. In *3rd International Conference on Human System Interaction*, pages 548–555. IEEE, 2010.

[34] Gabriel de Souza Pereira Moreira. Chameleon: a deep learning meta-architecture for news recommender systems. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 578–583, 2018.

[35] Magdalini Eirinaki, Jerry Gao, Iraklis Varlamis, and Konstantinos Tserpes. Recommender systems for large-scale social networks: A review of challenges and solutions, 2018.

[36] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[37] Flavio Figueiredo. On the prediction of popularity of trends and hits for user generated videos. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 741–746, 2013.

[38] Marko Gasparic and Andrea Janes. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software*, 113:101–113, 2016.

[39] R Stuart Geiger. Summary analysis of the 2017 github open source survey. *arXiv preprint arXiv:1706.02777*, 2017.

[40] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. Some from here, some from there: Cross-project code reuse in github. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 291–301. IEEE, 2017.

[41] GitHub. Discover repositories [online], November 2018. URL `https://github.com/dashboard/discover`.

[42] Jennifer Golbeck. The dynamics of web-based social networks: Membership, relationships, and change. *First monday*, 12(11), 2007.

[43] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[44] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296. ACM, 2016.

[45] Raf Guns and Ronald Rousseau. Recommending research collaborations using link prediction and random forest classifiers. *Scientometrics*, 101(2):1461–1473, 2014.

[46] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81. ACM, 2004.

[47] Jungpil Hahn, Jae Yun Moon, and Chen Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391, 2008.

[48] Caroline Haythornthwaite. Social networks and internet connectivity effects. *Information, Community & Society*, 8(2):125–147, 2005.

[49] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR Forum*, volume 51, pages 227–234. ACM New York, NY, USA, 2017.

[50] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[51] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Dec 2008. doi: 10.1109/ICDM.2008.22.

[52] Yan Hu, Shanshan Wang, Yizhi Ren, and Kim-Kwang Raymond Choo. User influence analysis for github developer social networks. *Expert Systems with Applications*, 108: 108–118, 2018.

[53] Lee Humphreys. Mobile social networks and social practice: A case study of dodgeball. *Journal of Computer-Mediated Communication*, 13(1):341–360, 2007.

[54] FO Isinkaye, YO Folajimi, and BA Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[55] Paul T Jaeger, Ben Shneiderman, Kenneth R Fleischmann, Jennifer Preece, Yan Qu, and Philip Fei Wu. Community response grids: E-government, social networks, and effective emergency management. *Telecommunications Policy*, 31(10):592–604, 2007.

[56] Niklas Jakob, Stefan Hagen Weber, Mark Christoph Müller, and Iryna Gurevych. Beyond the stars: exploiting free-text user reviews to improve the accuracy of movie recommendations. In *Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, pages 57–64, 2009.

[57] Oskar Jarczyk, Błażej Gruszka, Szymon Jaroszewicz, Leszek Bukowski, and Adam Wierzbicki. Github projects. quality analysis of open-source software. In *International Conference on Social Informatics*, pages 80–94. Springer, 2014.

[58] Jyun-Yu Jiang, Pu-Jen Cheng, and Wei Wang. Open source repository recommendation in social coding. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1173–1176. ACM, 2017.

[59] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.

[60] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In *Proceedings of the 13th International Workshop on Mining Software Repositories*, pages 291–302. ACM, 2016.

[61] Ron Kohavi and Dan Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *KDD*, pages 192–197, 1995.

[62] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[63] Miika Koskela, Inka Simola, and Kostas Stefanidis. Open source software recommendations using github. In *International Conference on Theory and Practice of Digital Libraries*, pages 279–285. Springer, 2018.

[64] Michael J Lee, Bruce Ferwerda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. Github developers use rockstars to overcome overflow of news. In

*CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138. ACM, 2013.

[65] Ting-Po Lee and Wen-Chien Chen. Applying link prediction for repository recommendation on github.

[66] Yangyang Li, Dong Wang, Haiyang He, Licheng Jiao, and Yu Xue. Mining intrinsic information by matrix factorization-based approaches for collaborative filtering in recommender systems. *Neurocomputing*, 249:48–63, 2017.

[67] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: Github as a collaborative social network. In *Icwsm*, 2014.

[68] Chao Liu, Dan Yang, Xiaohong Zhang, Baishakhi Ray, and Md Masudur Rahman. Recommending github projects for developer onboarding. *IEEE Access*, 2018.

[69] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[70] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[71] Zongyang Ma, Aixin Sun, and Gao Cong. On predicting the popularity of newly emerging hashtags in t witter. *Journal of the American Society for Information Science and Technology*, 64(7):1399–1410, 2013.

[72] Tadej Matek and Svit Timej Zebec. Github open source project recommendation system. *arXiv preprint arXiv:1602.02594*, 2016.

[73] John Mathieu, M Travis Maynard, Tammy Rapp, and Lucy Gilson. Team effectiveness 1997-2007: A review of recent advancements and a glimpse into the future. *Journal of management*, 34(3):410–476, 2008.

[74] Leo A. Meyerovich and Ariel S. Rabkin. Empirical analysis of programming language adoption. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*,

OOPSLA '13, pages 1–18, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2374-1. doi: 10.1145/2509136.2509515. URL http://doi.acm.org/10.1145/2509136.2509515.

[75] Koji Miyahara and Michael J Pazzani. Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International conference on artificial intelligence*, pages 679–689. Springer, 2000.

[76] Borja Moya-Gómez, María Henar Salas-Olmedo, Juan Carlos García-Palomares, and Javier Gutiérrez. Dynamic accessibility using big data: the role of the changing conditions of network congestion and destination attractiveness. *Networks and Spatial Economics*, 18(2):273–290, 2018.

[77] Khalil Muhammad, Aonghus Lawlor, Rachael Rafter, and Barry Smyth. Great explanations: Opinionated explanations for recommendations. In *International Conference on Case-Based Reasoning*, pages 244–258. Springer, 2015.

[78] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, volume 83. WoUongong, 1998.

[79] Naoki Orii. Collaborative topic modeling for recommending github repositories. 2012.

[80] Youngja Park, Zijie Qi, Suresh N Chari, and Ian M Molloy. Generating balanced classifier-independent training samples from unlabeled data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 266–281. Springer, 2012.

[81] C Michael Pilato, Ben Collins-Sussman, and Brian W Fitzpatrick. *Version Control with Subversion: Next Generation Open Source Version Control*. "O'Reilly Media, Inc.", 2008.

[82] Henrique Pinto, Jussara M Almeida, and Marcos A Gonçalves. Using early view patterns to predict the popularity of youtube videos. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 365–374, 2013.

[83] Lesandro Ponciano, Francisco Brasileiro, Robert Simpson, and Arfon Smith. Volunteers' engagement in human computation for astronomy projects. *Computing in Science & Engineering*, 16(6):52–59, 2014.

[84] Pearl Pu and Li Chen. Trust-inspiring explanation interfaces for recommender systems. *Knowledge-Based Systems*, 20(6):542–556, 2007.

[85] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[86] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 22. ACM, 2018.

[87] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[88] Marcel Rebouças, Renato O Santos, Gustavo Pinto, and Fernando Castor. How does contributors' involvement influence the build status of an open-source software project? In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 475–478. IEEE Press, 2017.

[89] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[90] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[91] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3): 56–58, March 1997. ISSN 0001-0782. doi: 10.1145/245108.245121. URL `http://doi.acm.org/10.1145/245108.245121`.

[92] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 0387858199, 9780387858197.

[93] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*, chapter 1, pages 1–35. Springer US, New York, NY, 2011. doi: 10.1007/978-0-387-85820-3.

[94] Martin Robillard, Robert Walker, and Thomas Zimmermann. Recommendation systems for software engineering. *IEEE software*, 27(4):80–86, 2010.

[95] Bin Shen, Xiaoyuan Su, Russell Greiner, Petr Musilek, and Corrine Cheng. Discriminative parameter learning of general bayesian network classifiers. In *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 296–305. IEEE, 2003.

[96] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding watchers on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 336–339. ACM, 2014.

[97] Parag Singla and Matthew Richardson. Yes, there is a correlation:-from social networks to personal behavior on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 655–664. ACM, 2008.

[98] Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. In *CHI'02 extended abstracts on Human factors in computing systems*, pages 830–831, 2002.

[99] Igor Steinmacher, Marco Aurélio Gerosa, and David Redmiles. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[100] Igor Steinmacher, Igor Scaliante Wiese, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. The hard life of open source software project newcomers. In *Proceedings of the 7th international workshop on cooperative and human aspects of software engineering*, pages 72–78. ACM, 2014.

[101] Ján Suchal and Pavol Návrat. Full text search engine as scalable k-nearest neighbor recommendation system. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 165–173. Springer, 2010.

[102] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[103] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 141–150. IEEE, 2015.

[104] Poonam B Thorat, RM Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.

[105] Ferdian Thung, Tegawende F Bissyande, David Lo, and Lingxiao Jiang. Network structure of social coding in github. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 323–326. IEEE, 2013.

[106] Jeffrey Travers and Stanley Milgram. The small world problem. *Phychology Today*, 1(1):61–67, 1967.

[107] Oren Tsur and Ari Rappoport. What's in a hashtag? content based prediction of the spread of ideas in microblogging communities. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 643–652, 2012.

[108] Saúl Vargas. New approaches to diversity and novelty in recommender systems. In *Fourth BCS-IRSG symposium on future directions in information access (FDIA 2011), Koblenz*, volume 31, 2011.

[109] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Perceptions of diversity on github: A user survey. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 50–56. IEEE Press, 2015.

[110] Noah Webster and Noah McKechnie. *Webster's new twentieth century dictionary of yhe english language unabrid ged*. Number C/423 W4. 1975.

[111] Yandi Xia, Giuseppe Di Fabbrizio, Shikhar Vaibhav, and Ankur Datta. A content-based recommender system for e-commerce o ers and coupons. 2017.

[112] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th international conference on software engineering*, pages 419–429. IEEE Computer Society, 2003.

[113] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. Reviewer recommender of pull-requests in github. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 609–612. IEEE, 2014.

[114] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: Determinants of pull request evaluation latency on github. In *Mining software repositories (MSR), 2015 IEEE/ACM 12th working conference on*, pages 367–371. IEEE, 2015.

[115] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218, 2016.

[116] Quan Yuan, Shiwan Zhao, Li Chen, Yan Liu, Shengchao Ding, Xiatian Zhang, and Wentao Zheng. Augmenting collaborative recommender by fusing explicit social relationships. In *Workshop on Recommender Systems and the Social Web, Recsys 2009*, 2009.

[117] Lingxiao Zhang, Yanzhen Zou, Bing Xie, and Zixiao Zhu. Recommending relevant projects via user behaviour: an exploratory study on github. In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, pages 25–30. ACM, 2014.

[118] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. Detecting similar repositories on github. In *Software Analysis, Evolution and*

*Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 13–23. IEEE, 2017.

[119] Lei Zheng. A survey and critique of deep learning on recommender systems. *no. September*, page 31, 2016.

[120] Xujuan Zhou, Yue Xu, Yuefeng Li, Audun Josang, and Clive Cox. The state-of-the-art in personalized recommender systems for social networking. *Artificial Intelligence Review*, 37(2):119–132, 2012.

# Apêndice A

# Comparação de *F-Measure*

Na Tabela A.1 temos a comparação dos valores de *F-Measure* na investigação para QP1, QP2 e QP3.

Tabela A.1: Melhores Resultados dos Modelos para F-Measure para QP1 (FM-1), QP2 (FM-2) e QP3 (FM-3)

| Dataset | | Aprendizado da Classificação | | |
|---|---|---|---|---|
| | | FM-1 | FM-2 | FM-3 |
| Item-KNN | top-3 | 0.00043 | 0.00043 | 0.000009 |
| | top-5 | 0.00038 | 0.00038 | 0.000006 |
| | top-10 | 0.00034 | 0.00034 | 0.000003 |
| | top-50 | 0.00062 | 0.00022 | 0.0000007 |
| | top-100 | 0.00198 | 0.00036 | 0.0000003 |
| RF | top-3 | 0.00066 | 0.00008 | 0.4674 |
| | top-5 | 0.00092 | 0.00010 | 0.3045 |
| | top-10 | 0.00143 | 0.00022 | 0.1626 |
| | top-50 | 0.00462 | 0.00053 | 0.0343 |
| | top-100 | 0.00631 | 0.00095 | 0.0172 |
| XGBoost | top-3 | 0.00331 | 0.00529 | 0.4620 |
| | top-5 | 0.00391 | 0.00552 | 0.2997 |
| | top-10 | 0.00492 | 0.00669 | 0.1594 |
| | top-50 | 0.00516 | 0.00785 | 0.0335 |
| | top-100 | 0.00631 | 0.00994 | 0.0168 |
| MF | top-3 | 0.00010 | 0.00010 | 0.4999 |
| | top-5 | 0.00006 | 0.00006 | 0.3333 |
| | top-10 | 0.00005 | 0.00006 | 0.1818 |
| | top-50 | 0.00011 | 0.00011 | 0.0392 |
| | top-100 | 0.00025 | 0.00015 | 0.0197 |