

**Um Modelo de Classificação de
Documentação para Novatos
em Projetos de Software Livre**

Luiz Felipe Fronchetti Dias

DISSERTAÇÃO APRESENTADA AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO TÍTULO DE
MESTRE EM CIÊNCIAS

Programa: Ciência da Computação

Orientador: Prof. Dr. Marco Aurélio Gerosa

Coorientador: Prof. Dr. Igor Steinmacher

Durante o desenvolvimento deste trabalho o autor
recebeu auxílio financeiro da FAPESP (18/02596-1)

São Paulo
01 de Janeiro de 2023

**Um Modelo de Classificação de
Documentação para Novatos
em Projetos de Software Livre**

Luiz Felipe Fronchetti Dias

Esta é a versão original da dissertação
elaborada pelo candidato Luiz
Felipe Fronchetti Dias, tal como
submetida à Comissão Julgadora.

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Resumo

Luiz Felipe Franchetti Dias. **Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Contexto: Projetos de software livre representam uma grande parcela do mercado de desenvolvimento de software. Neste contexto, uma diversidade de projetos de software livre contam com contribuições voluntárias para se manterem ativos. Embora importantes para o ecossistema dos projetos, novos contribuidores ao tentarem ingressar voluntariamente em projetos de software livre, tendem a enfrentar um conjunto de barreiras que dificultam o processo de contribuição. Tais barreiras levam novatos a desistirem de contribuir com projetos de software livre. Entre as dificuldades já constatadas, existem barreiras associadas às documentações dos projetos que, muitas vezes, não atendem as necessidades que novos contribuidores têm durante o processo de contribuição.

Objetivo: Almejando contribuir com a avaliação de documentações de projetos de software livre, esta pesquisa se baseia na construção de um modelo de classificação, capaz de identificar trechos em arquivos de contribuição de projetos de software livre que sejam relevantes a novatos durante o processo de entrada em tais projetos. Acreditamos que tal ferramenta possa auxiliar novos contribuidores a compreender aspectos essenciais dos projetos aos que eles almejam contribuir, bem como permitir uma melhor avaliação dos projetos quanto à qualidade das documentações produzidas para novatos.

Método: Um modelo de classificação de trechos de documentação relevantes a novos contribuidores foi implementado a partir da análise qualitativa de arquivos de contribuição encontrados em repositórios de software livre. Utilizando como base algoritmos de aprendizado, este modelo recebe como entrada parágrafos de documentações de projetos, e identifica para cada um dos parágrafos, uma entre seis categorias de informação relevantes a novos contribuidores. O modelo de classificação, bem como os dados e resultados obtidos durante a execução deste método, foram analisados sob diferentes perspectivas a partir de duas etapas, uma de avaliação estatística, e outra através de um questionário aplicado à desenvolvedores de software com e sem experiência em software livre.

Resultados:

Palavras-chave: Software Livre e de Código Aberto. Novatos. Análise de Documentação.

Abstract

Luiz Felipe Franchetti Dias. **Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre**. Thesis (Masters). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Context: Open source projects constitute a significant part of the software development industry. In this context, for projects to remain active, voluntary contributions are expected. However, new contributors, when voluntarily trying to join open source projects, often face a set of barriers that compromises the contribution process. Such barriers tend to lead newcomers to give up contributing to these projects and, for this reason, are widely studied in the literature. Among the difficulties already noted in the literature, there are barriers associated with the documentation of open source projects that often do not meet the needs that new contributors have during the contribution process.

Objective: Aiming to contribute to the documentation of open source projects, we propose the development of a classification model, capable of identifying snippets in open source projects documentation that are relevant to newcomers. This tool, at the end of this study, will help newcomers to understand essential aspects of the projects they aim to contribute, mitigating potential difficulties faced during the contribution process.

Method: From an experiment to be conducted with newcomers to open source projects, and researchers in software engineering, a model of classification of categories relevant to new contributors will be built. Using machine learning concepts as a fundamental basis, this model will receive as input project documentation paragraphs, and should identify for each paragraph, a set of categories of information relevant to new contributors, which will be defined at a preliminary stage. The classification model, as well as the data and results obtained during the execution of this method, will be analyzed from different perspectives in a statistical evaluation stage, and through questionnaires, to be applied with newcomers and core contributors in the context of open source software.

Expected Results: We hope with this study to answer whether it is possible to identify relevant documentation snippets to new open source contributors using classification algorithms, and to present which features are most relevant for identifying these snippets, and the acceptance of predictions by contributors in this context.

Keywords: Free/Libre and Open Source Software. Newcomers. Documentation Analysis.

Lista de Abreviaturas

FLOSS	Free/Libre Open Source Software
NLP	Natural Language Processing
API	Application Programming Interface
YODA	Young NewcOmer Developer Assistant
CODES	Mining SourCe COde Descriptions from DevelopErs DiScussions

Lista de Figuras

1.1	Padrões de entrada de novatos encontrados em projetos de software livre	2
2.1	Estrutura em camadas de uma comunidade de software livre	7
2.2	Modelo de barreiras enfrentadas por novatos em projetos de software livre	9
2.3	Exemplo de projeto cadastrado no <i>Up For Grabs</i>	12
2.4	Exemplo de documentação gerada pela ferramenta Javadoc	14
3.1	Método de pesquisa	16
3.2	Número de parágrafos analisados por arquivo.	20
3.3	Número de projetos contendo cada categoria de informação.	20
3.4	Número de categorias de informação atendidas por projeto.	22
3.5	Construção do modelo de classificação	24
3.6	Exemplo de questão aplicada no questionário.	28
4.1	Matriz de confusão para o modelo de classificação final (LinearSVC). . . .	31
4.2	Características com maior importância para predição de cada categoria de informação.	33
4.3	Avaliação das predições feitas pelo modelo de classificação final, de acordo com participantes do questionário.	35
4.4	Número de projetos preditos contendo cada categoria de informação. . .	37
4.5	Número de categorias de informação atendidas por projeto predito. . . .	38

Lista de Tabelas

3.1	Categorias estabelecidas para identificação nas documentações dos projetos.	17
3.2	Número de projetos removidos por linguagem e as razões para exclusão. O valor “n” representa o total de projetos extraídos para determinada linguagem. É válido mencionar que certos arquivos podem ter sido removidos por mais de uma razão.	18
3.3	Exemplos de características heurísticas utilizadas.	23
4.1	Valores de f-measure para os classificadores testados no processo de validação cruzada.	30
4.2	Performance do modelo de classificação final (LinearSVC).	30
4.3	Performance do modelo de classificação características específicas.	32
4.4	Experiência em programação e em software livre dos participantes do questionário.	34

Sumário

1	Introdução	1
1.1	Estudo preliminar	2
1.2	Questões de pesquisa	3
1.3	Organização do trabalho	3
2	Referencial Teórico	5
2.1	Software Livre	5
2.1.1	Representatividade	6
2.1.2	Funcionamento	6
2.1.3	Organização	7
2.2	Barreiras enfrentadas por novatos	7
2.3	Problemas em documentações de software	10
2.4	Trabalhos Relacionados	11
2.4.1	Apoio a novatos em software livre	11
2.4.2	Suporte a documentação de software	13
3	Metodologia	15
3.1	Método	15
3.1.1	Definição das categorias	15
3.1.2	Extração dos dados	16
3.1.3	Análise dos dados	19
3.1.4	Descrição dos dados analisados	19
3.1.5	Pré-processamento	21
3.1.6	Classificação	23
3.1.7	Avaliação do modelo de classificação	26
4	Resultados da classificação	29
4.1	Escolha do modelo de classificação	29
4.2	Avaliação do modelo de classificação	30

4.3	Avaliação das características	32
4.3.1	Comparação entre conjuntos de características	32
4.3.2	Características com maior importância	32
4.4	Questionário sobre predições	34
4.5	Avaliação das documentações restantes	36
5	Ameaças à validade	39
5.0.1	Generalidade dos resultados	39
5.0.2	Representatividade da perspectiva dos contribuidores	39
5.0.3	Cobertura das categorias de informação	39
6	Conclusão	41
6.1	Discussão	41
6.1.1	Falta de informação essencial para novatos	41
6.1.2	A maioria dos arquivos são focados nas etapas finais da contribuição	41
6.2	Considerações finais	41
	Referências	43

Capítulo 1

Introdução

Projetos de software livre estão presentes em várias áreas do desenvolvimento de software, atendendo a uma parcela significativa do mercado consumidor de programas de computador (BONACCORSI e ROSSI, 2003). Muitos projetos neste contexto, para se manterem ativos, contam com contribuições voluntárias de desenvolvedores de código (CROWSTON *et al.*, 2012). Voluntários, ao tentarem ingressar em projetos de software livre, costumam sofrer uma série de dificuldades, que comprometem a realização de contribuições (I. STEINMACHER, WIESE *et al.*, 2014). Tais dificuldades levam, muitas das vezes, novatos a desistirem de contribuir com projetos de software livre, refletindo no enfraquecimento das comunidades.

Entre as dificuldades enfrentadas por novatos, se encontram barreiras associadas a documentação dos projetos de software livre que, muitas das vezes, se encontra obsoleta, incompleta, incoerente ou inconsistente (BRIAND, 2003; AGHAJANI *et al.*, 2019). Além do mais, evidências mostram que em muitos casos, as documentações de projetos de software livre, não atendem as necessidades que novos contribuidores têm (I. STEINMACHER, WIESE *et al.*, 2014). Tais problemas de documentação, tendem a gerar confusão naqueles que desejam realizar uma primeira contribuição e por esta razão precisam ser solucionados.

Para mitigar problemas de documentações de software, uma série de ferramentas são propostas por cientistas e desenvolvedores. ZHONG e SU (2013), por exemplo, sugerem um mecanismo de identificação de erros em documentações, utilizando uma combinação de técnicas de processamento de linguagem natural para encontrar trechos de documentação em códigos fonte que estejam incorretos ou obsoletos. Outro exemplo de trabalho é proposto por HAIDUC *et al.* (2010), que estabelece um mecanismo de geração automática de documentação para classes e métodos em Java, visando aprimorar a compreensão de códigos fonte. Até mesmo fora do contexto acadêmico, uma série de projetos relacionados à documentação de software também são propostos, entre eles, o Javadoc¹, que analisa anotações e comentários de código e gera automaticamente páginas de documentação em formato HTML para os códigos analisados.

Apesar destes trabalhos contribuírem para documentação de software, nenhuma das ferramentas propostas foca em novatos em software livre. Por esta razão, implementa-

¹docs.oracle.com/javase/10/javadoc

mos neste estudo a criação de um modelo de classificação que identifique trechos em documentações de projetos de software livre que sejam relevantes a novos contribuidores. Acreditamos que este modelo de classificação deverá não só auxiliar novatos a encontrarem trechos em documentações que sejam úteis durante o processo de contribuição em software livre, como também poderá contribuir com a avaliação da qualidade das documentações já existentes em projetos neste contexto.

1.1 Estudo preliminar

Este estudo dá sequência a uma pesquisa preliminar (FRONCHETTI *et al.*, 2019), na qual foram investigadas características que influenciam a entrada de novatos em projetos de software livre. Nesta pesquisa preliminar, 15 características de 450 projetos de software livre foram extraídas e analisadas, entre elas, a idade, o domínio de aplicação e o número de linguagens de programação utilizadas em cada um dos projetos selecionados. O algoritmo de clusterização K-Spectral Centroid (KSC) foi utilizado para investigar a taxa de crescimento de novatos nos projetos, e três diferentes padrões de entrada de novatos foram encontrados: logarítmico, exponencial e linear.

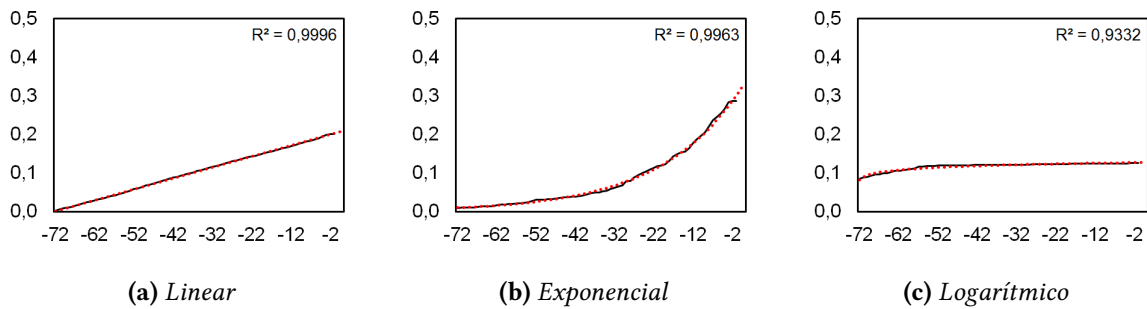


Figura 1.1: Padrões de entrada de novatos encontrados em projetos de software livre (FRONCHETTI *et al.*, 2019).

Com base nesses padrões de crescimento, um modelo de classificação foi construído a fim de compreender quais fatores poderiam explicar as taxas de entrada de novatos em cada um dos projetos. Foi constatado que fatores como a popularidade dos projetos, o tempo para revisão de submissões, a idade e as linguagens de programação utilizadas nos repositórios de código eram os fatores que melhor explicavam a entrada de novatos no projetos de software livre analisados.

Também foram avaliados fatores dicotômicos associados à documentação dos projetos, tais como a existência de arquivos README e CONTRIBUTING nos repositórios de código fonte, conhecidos por serem fornecedores primários de informação para novatos em software livre. Nenhuma das características relacionadas à documentação dos projetos apresentaram grande influência na atratividade de novatos. A existência de arquivos contendo o código de conduta dos projetos, por exemplo, foi o fator que apresentou a menor influência entre todos os analisados.

Essa baixa influência pode ser considerada uma ameaça à validade deste estudo preliminar, já que as características relacionadas às documentações foram avaliadas de maneira dicotômica, considerando apenas o fato de um determinado arquivo existir ou não. Por

este motivo, surge a necessidade deste novo estudo, cujo objetivo é identificar trechos de documentações que sejam relevantes aos novos contribuidores. Identificar tais trechos, não só pode abrir espaço para futuras ferramentas que auxiliarão novatos, como também pode contribuir para avaliar a qualidade das documentações que projetos de software livre disponibilizam.

1.2 Questões de pesquisa

A metodologia proposta para este estudo é fundamentada a partir de três questões pesquisa, focadas em compreender e avaliar o modelo de classificação construído:

QP1 É possível identificar trechos em documentações de projetos de software livre que sejam relevantes a novos contribuidores utilizando algoritmos de classificação?

QP2 Como diferentes características influenciam as predições feitas pelo classificador de trechos de documentação?

QP3 Qual a percepção dos desenvolvedores de software livre em relação à identificação de trechos de documentação relevantes a novos contribuidores?

1.3 Organização do trabalho

Este projeto de dissertação está organizado em quatro capítulos. O Capítulo 2 define o conceito de software livre, discutindo barreiras que novatos enfrentam neste contexto, problemas usualmente encontrados em documentação de software, e trabalhos relacionados que buscam solucionar problemas similares aos encontrados nesta pesquisa. O Capítulo 3 apresenta a metodologia proposta para este estudo, descrevendo um conjunto de cinco passos que guiaram toda a pesquisa. Em seguida, o Capítulo 4 apresenta os resultados obtidos com a construção do modelo de classificação proposto, bem como responde as questões de pesquisa definidas. Por fim, o Capítulo 6 apresenta uma conclusão para o trabalho realizado.

Capítulo 2

Referencial Teórico

Como o objetivo desta pesquisa concentra-se em investigar por meio de aprendizado de máquina a documentação de projetos de software livre, a revisão bibliográfica foi dividida em cinco seções: A Seção 2.1 traz uma breve explicação sobre o que é software livre e como desenvolvedores trabalham e se organizam neste contexto. Na Seção 2.2, é apresentado o conceito de barreiras enfrentadas por novatos em software livre. Na Seção 2.3, são apresentados problemas associados à documentação de projetos de software, dando ênfase naqueles constatados no contexto de software livre. Por fim, na Seção 2.4, são apresentados os trabalhos relacionados a este estudo.

2.1 Software Livre

Software livre (do inglês, *free software*) é o tipo de software que em sua essência é distribuído livremente, acompanhado por uma licença de software, e que conta com a disponibilização de seu código-fonte em algum meio acessível a usuários e desenvolvedores (Cristina GACEK e Budi ARIEF, 2004). Um programa de computador para ser considerado livre, deve fornecer a quem o utiliza quatro liberdades essenciais (STALLMAN, 2002):

- A liberdade de executar o programa, para qualquer propósito;
- A liberdade de estudar como o programa funciona, e poder adaptá-lo;
- A liberdade de redistribuir cópias do programa a outros usuários;
- A liberdade de aperfeiçoar o programa, e poder liberar os aperfeiçoamentos a comunidade.

Outro termo geralmente associado a software livre é o conceito de software de código aberto (do inglês, *open source software*), que apresenta certas diferenças¹ em sua definição se comparado a software livre. Um possível antônimo para software livre é o software proprietário, que restringe legalmente a terceiros o acesso ao código fonte do programa desenvolvido (FUGGETTA, 2003).

¹gnu.org/philosophy/open-source-misses-the-point.html

2.1.1 Representatividade

Embora o termo software livre ainda seja desconhecido por muitos usuários, projetos neste contexto têm mostrado grande influência no cenário de desenvolvimento e consumo de software (HÖST e ORUČEVIĆ-ALAGIĆ, 2011). O servidor Apache é um exemplo de projeto de software livre popularmente conhecido entre desenvolvedores ao redor do mundo. Mantido por voluntários, o Apache hospedava em Julho de 2019², mais de 44% dos websites ativos na internet. O código fonte principal do sistema operacional Android é outro exemplo de projeto de software livre que tem apresentado grande adesão de usuários no mercado de software, podendo ser considerado um dos maiores sistemas operacionais para dispositivos móveis do mundo, representou em Setembro de 2019 uma parcela de mais de 76%³ de adesão em dispositivos móveis.

Não apenas usuários e desenvolvedores têm demonstrado interesse em software livre, evidências mostram que diversas empresas também se encontram alinhadas a esta mesma filosofia de liberdade (PINTO, Igor STEINMACHER *et al.*, 2018). A linguagem de programação Swift⁴, desenvolvida pela Apple, é um exemplo de programa de computador que por muito tempo foi proprietário, e que hoje faz parte do contexto de software livre. Até mesmo a Microsoft, reconhecida por sua popularidade com projetos de código fechado como Windows e Office, lançou em Novembro de 2015 uma versão aberta do seu editor de texto, o Visual Studio Code⁵. Essas companhias parecem não só estarem interessadas em tornar disponíveis publicamente o código fonte de seus projetos, como também tem mostrado interesse em contribuir com o cenário de software livre, pagando desenvolvedores para trabalharem em projetos neste contexto (PINTO, DIAS *et al.*, 2018).

2.1.2 Funcionamento

Geralmente, um novo projeto de software livre se inicia quando um programador possui uma ideia, problema ou porção de código a ser desenvolvido (RAYMOND, 1999). O programador submete uma proposta a comunidades colaborativas de software, anunciando seu projeto a outros desenvolvedores. Tais desenvolvedores, quando atraídos pelo projeto proposto, tendem a buscar por informações a respeito do projeto, e passam a participar do desenvolvimento dos códigos. Com o tempo e com a adesão de novos contribuidores, uma nova comunidade de software livre se forma em torno do projeto. Plataformas de codificação passam a ser utilizadas como meio de gerenciamento do código fonte, e canais de comunicação são estabelecidos entre os membros da comunidade.

Projetos neste contexto usualmente são mantidos a partir de contribuições de desenvolvedores internos e externos às comunidades de software livre. Para contribuir com um projeto, um desenvolvedor externo adquire uma cópia do código fonte através de uma plataforma de codificação, implementa modificações e submete as mudanças novamente ao repositório de código do projeto (Audris Mockus *et al.*, 2000). As modificações são revisadas por um grupo de mantenedores e, quando estão de acordo com os critérios de aceitação, são anexadas aos binários do programa. Desenvolvedores interessados em

²w3techs.com/technologies/overview/web_server/all

³<https://gs.statcounter.com/os-market-share/mobile/worldwide>

⁴developer.apple.com/swift/blog/?id=34

⁵github.com/microsoft/vscode/issues/60

participar de uma comunidade de software livre tendem a construir sua reputação por meio de contribuições ao código fonte dos programas desenvolvidos, de onde com o tempo, passam a ganhar espaço e notoriedade (KROGH *et al.*, 2003).

2.1.3 Organização

Em relação à organização dos projetos, NAKAKOJI *et al.* (2002) sintetizam a estrutura social de comunidades de software livre a partir de um modelo de camadas. Segundo os autores, nas camadas mais internas se encontram os principais desenvolvedores, aqueles que geralmente contribuem com a maior parte do código fonte e são responsáveis por revisar futuras contribuições. Ao meio da estrutura, se encontram membros do projeto que são responsáveis por contribuir com pequenas correções e aprimoramentos. E por fim, nas camadas mais externas, se encontram diferentes níveis de usuários, que não contribuem com o código fonte do projeto, mas que eventualmente relatam problemas e dão sugestões, além de desfrutar dos recursos oferecidos pelo programa. A representação do modelo de camadas é apresentada na Figura 2.1.

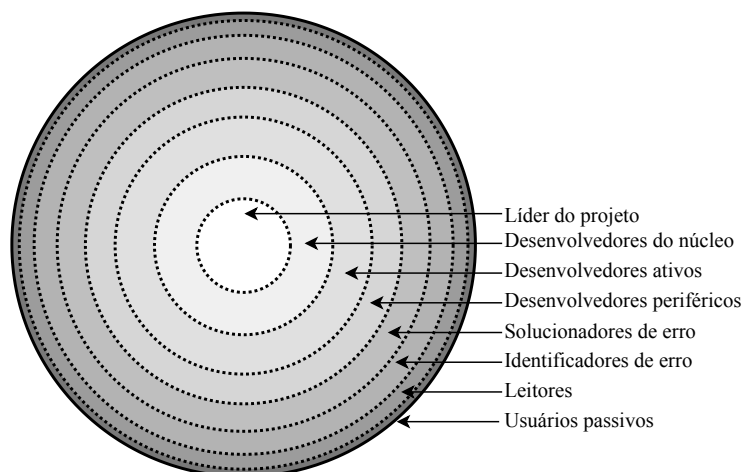


Figura 2.1: Estrutura em camadas de uma comunidade de software livre (NAKAKOJI *et al.*, 2002)³

Além dos grupos apresentados pelo modelo de NAKAKOJI *et al.* (2002), existem ainda os novos contribuidores que, apesar de não necessariamente constituírem parte da hierarquia de um projeto, por serem considerados essenciais ao desenvolvimento dos projetos, são amplamente estudados pela literatura em software livre. Neste contexto, um novo contribuidor, novato ou recém-chegado, pode ser definido como um desenvolvedor interessado em ingressar em uma comunidade de software livre por meio de contribuições. Entre as razões pelas quais novatos são amplamente estudados pela literatura, se encontram as dificuldades que estes enfrentam ao tentarem ingressar em projetos de software livre, como mostra a seção a seguir.

2.2 Barreiras enfrentadas por novatos

Como as contribuições em software livre costumam ser voluntárias, o ato de atrair novos contribuidores para tais comunidades é considerado fator essencial para continuidade

de muitos projetos (QURESHI e FANG, 2011). Como em qualquer atividade coletiva, se membros ao saírem de uma comunidade não forem substituídos por novos contribuidores, a comunidade irá eventualmente deixar de existir, já que novatos podem ser considerados fonte de inovação, ideias e trabalho (KRAUT *et al.*, 2012).

No entanto, atrair novos contribuidores para comunidades de software livre não é uma tarefa trivial (ZHOU e A. MOCKUS, 2015). Em um estudo sobre empresas que compartilham comunidades neste contexto, DAHLANDER e MAGNUSSON (2008) mostram que até mesmo para comunidades bem estabelecidas como o MySQL, diversos problemas são enfrentados ao atrair novos contribuidores. Entre os problemas que justificam a baixa adesão de novatos a projetos de software livre estão um conjunto de barreiras enfrentadas por novatos durante o processo de entrada e integração a tais projetos.

KROGH *et al.* (2003) descrevem as barreiras enfrentadas por novatos como um conjunto de dificuldades, empecilhos ou deficiências no processo de contribuição que atrapalham desenvolvedores a realizarem suas primeiras colaborações. De acordo com os autores, as barreiras estão intrinsecamente relacionadas ao nível de especialização de cada desenvolvedor. Como a complexidade de um projeto tende a crescer conforme seu desenvolvimento, ingressar em meio à sua execução tende a ser um processo lento e dificultoso a novos desenvolvedores que, por muitas vezes, desconhecem os processos e tecnologias utilizadas pela comunidade.

HANNEBAUER e GRUHN (2017) dividem barreiras enfrentadas por novatos em duas categorias, submissão e modificação. De acordo com os autores, novatos tendem a enfrentar dificuldades tanto durante a elaboração de uma nova contribuição, como durante o processo de submissão do código modificado ao projeto. Em uma perspectiva complementar, I. STEINMACHER, WIESE *et al.* (2014) mostram que tais barreiras não somente estão associadas às motivações e características do desenvolvedor ao contribuir com um projeto, como também são desencadeadas por problemas da própria comunidade à qual ele almeja participar. Problemas como falta de documentação e a má recepção fornecida a novos contribuidores são exemplos de problemas que podem impactar a entrada dos novatos em comunidades de software livre.

Entre os resultados dos estudos que sintetizam as dificuldades enfrentadas por novatos em software livre se encontra o modelo de barreiras proposto por I. STEINMACHER, CHAVES *et al.* (2014). Dividido em seis categorias, o modelo apresenta cinquenta e oito barreiras enfrentadas por novos contribuidores. Entre as categorias, foram constatadas dificuldades relacionadas ao processo de recepção dos novatos, às características dos próprios desenvolvedores, à falta de orientação durante o processo de contribuição, aos problemas com documentação, aos problemas associados às diferenças culturais e aos obstáculos técnicos. A Figura 2.2 apresenta uma tradução livre do modelo de barreiras proposto pelos autores. Vale ressaltar, que tal modelo serviu de inspiração para elaboração de categorias relevantes a novatos em software livre utilizadas neste estudo, e apresentadas na Tabela 3.1.

O fato de tais comunidades não atraírem novos contribuidores o suficiente usualmente implica na falta de condições necessárias para se manter o desenvolvimento dos projetos ativo, levando, por consequência, muitas comunidades de software livre a inatividade. Por esta razão, é tido como proposta deste estudo a elaboração de uma metodologia que contribua com problemas que novatos enfrentam em relação às documentações de projetos

2.2 | BARREIRAS ENFRENTADAS POR NOVATOS

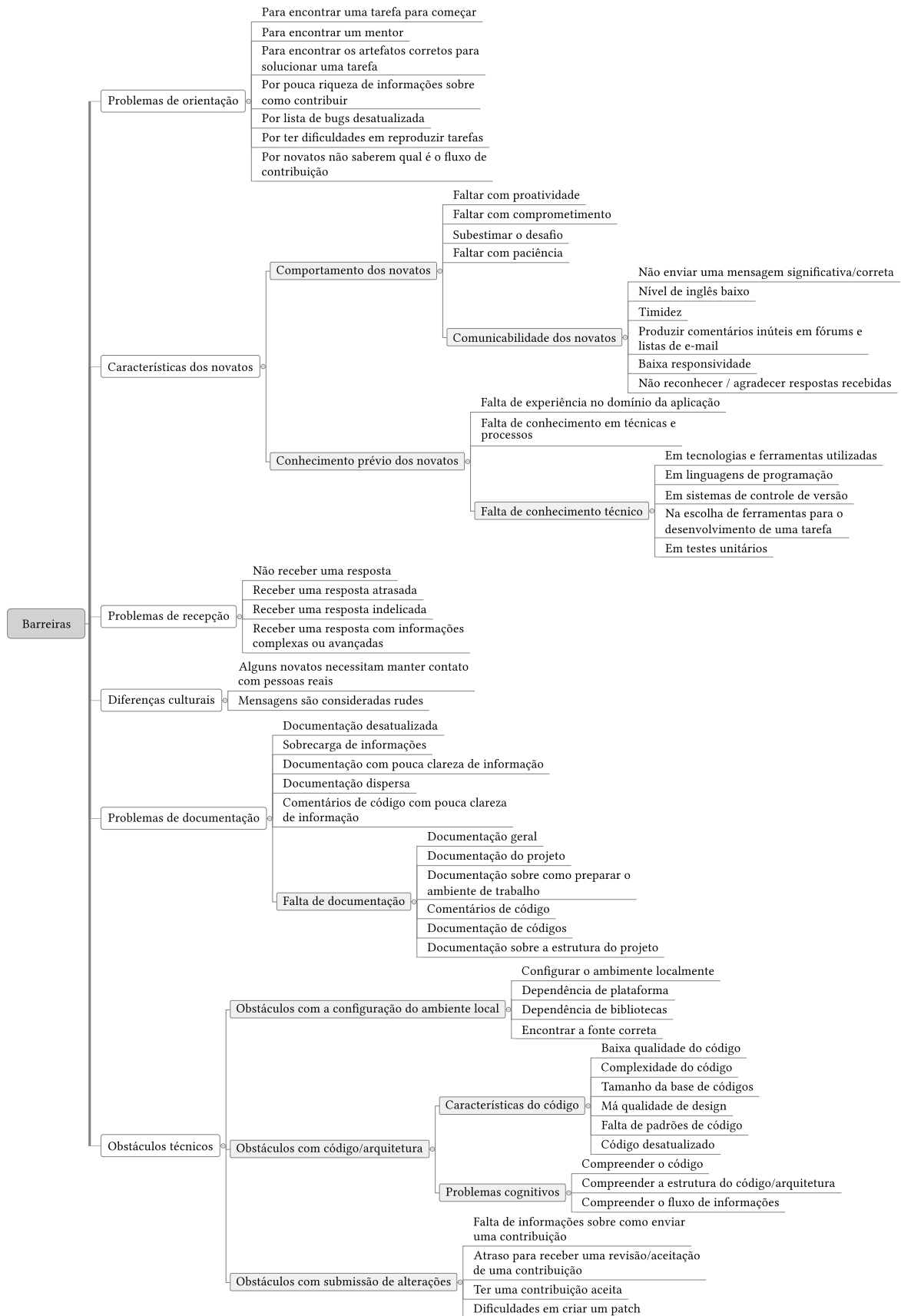


Figura 2.2: Modelo de barreiras enfrentadas por novatos em projetos de software livre (I. STEINMA-CHER, CHAVES et al., 2014).

de software livre, de modo com que algumas das barreiras apresentadas por I. STEINMACHER, CHAVES *et al.* (2014), e demais trabalhos relacionados, sejam reduzidas ou deixem de existir. Na seção a seguir, uma visão geral sobre problemas enfrentados em documentações de software é apresentada.

2.3 Problemas em documentações de software

A documentação de um programa de computador, quando correta, completa e consistente, tende a ser uma fonte crucial de informações para desenvolvedores (KAJKO-MATTSSON, 2005). O propósito de se documentar um programa é contribuir com a preservação dos processos desenvolvidos durante sua elaboração, de modo com que futuras modificações possam ser feitas (AGHAJANI *et al.*, 2019). Ainda que a elaboração de uma documentação possa ser custosa, a preservação do entendimento de um sistema pode apresentar diversos benefícios ao seu desenvolvimento, tendo reflexo, por exemplo, na produtividade dos desenvolvedores, na redução dos custos de manutenção e na extensão da vida útil do sistema desenvolvido (ZHI *et al.*, 2015; SOUZA *et al.*, 2005).

No entanto, apesar dos benefícios, evidências mostram que documentações de programas de computador costumam apresentar uma diversidade de problemas (BRIAND, 2003). Em muitos casos, as documentações se encontram incorretas, obsoletas, incompletas ou incoerentes, o que pode comprometer o entendimento do programa por parte dos desenvolvedores (AGHAJANI *et al.*, 2019; FORWARD e LETHBRIDGE, 2002). Problemas de documentação não se restringem a uma única área de aplicação e podem estar presentes em diversos contextos do desenvolvimento de software.

Ao investigar problemas em documentações de API's, UDDIN e ROBILLARD (2015) dividem tais problemas em duas categorias, conteúdo e apresentação. Documentações obsoletas, incoerentes ou incompletas são exemplos de problemas associados à categoria conteúdo, que visa compreender problemas relacionados às informações contidas em cada documentação. A divisão desnecessária do texto em páginas ou seções e o excesso de informações sobre um único tópico ou elemento são exemplos de problemas associados à categoria apresentação, que abrange a estrutura e o modo como as documentações são expostas a desenvolvedores.

Além das categorias mencionadas, ROBILLARD (2009) apresenta outras três classes de problemas associados a documentações de software: experiência, ambiente técnico e processo. Na classe experiência, o autor identifica que muitos dos obstáculos enfrentados com documentações de software são provenientes da falta de experiência prévia do desenvolvedor, que muitas vezes desconhece os conceitos fundamentais do sistema e que, por consequência, passa a enfrentar problemas com as documentações. Os diferentes ambientes de desenvolvimento nos quais um programador trabalha são apresentados como outro fator gerador de obstáculos em documentações de software, que muitas vezes costumam apresentar informações apenas para um único tipo de sistema. Por fim, o tempo e a maneira como os desenvolvedores se dedicam a compreender uma determinada documentação também são apresentados como fatores geradores de obstáculos, já que influenciam indiretamente no processo de entendimento da documentação por parte dos desenvolvedores.

No contexto de software livre, onde a documentação dos projetos é crucial para integração de novatos às comunidades, diversos problemas também são enfrentados (C. GACEK e B. ARIEF, 2004; ABERDOUR, 2007). Além de problemas associados às categorias mencionadas anteriormente, I. STEINMACHER, CHAVES *et al.* (2014) identificam em seu modelo de barreiras outros tipos de problemas com documentação, tais como a falta de clareza nas informações dispostas, a dispersão das informações sobre diferentes meios de comunicação e a sobrecarga de informações a respeito de um mesmo tópico.

2.4 Trabalhos Relacionados

Nesta seção, são apresentados trabalhos que buscam solucionar problemas similares aos atacados neste estudo, incluindo o apoio a entrada de novatos em software livre e a automatização da busca por trechos em documentação de software. Para cada tópico a seguir, são apresentados trabalhos encontrados na literatura e trabalhos provenientes de fora do contexto acadêmico.

2.4.1 Apoio a novatos em software livre

Na literatura, MALHEIROS *et al.* (2012) propõem uma ferramenta denominada *Mentor* que, através de algoritmos de recomendação, sugere a novos contribuidores arquivos do código fonte de projetos de software livre que podem contribuir com a realização de uma determinada tarefa. Tal ferramenta, facilita a busca de arquivos que estejam associados à, por exemplo, solução de um erro no programa desenvolvido. Outra ferramenta similar é proposta por CUBRANIC *et al.* (2005), que também utiliza de algoritmos de recomendação para sugerir a novos contribuidores arquivos do código fonte de projetos que possam contribuir na execução de uma tarefa. A diferença é que nesta ferramenta não são sugeridos apenas arquivos de código fonte, mas também trechos provenientes de outras fontes, como documentações dos projetos, canais de comunicação, relatórios de erro e planos para teste.

Buscando auxiliar novatos na escolha de uma primeira tarefa, WANG e SARMA (2011) propõem um mecanismo inteligente de busca por erros não resolvidos em projetos de software livre. De acordo com os autores, uma boa maneira de ingressar em um projeto de software livre se encontra na resolução de erros do programa desenvolvido. Encontrar erros que se adéquem às características técnicas e de interesse do novo contribuidor é um fator que deve ser levado em consideração, já que pode ser primordial na integração do novato a projetos neste contexto. O sistema proposto tem acesso à lista de erros de um projeto e possibilita que desenvolvedores encontrem erros que desejam solucionar por meio de buscas inteligentes, além de identificar erros similares àqueles que os novatos demonstrem interesse.

Engajados com a orientação de novatos em software livre, CANFORA *et al.* (2012) descrevem uma abordagem de auxílio a novatos denominada YODA⁶, que objetiva identificar e recomendar mentores para novos contribuidores em projetos de software livre. Por meio de algoritmos de recomendação e utilizando dados extraídos de listas de e-mail e sistemas

⁶spanichella.github.io/tools.html#yoda

de controle de versão, o YODA recomenda um mentor através de uma análise sistemática do histórico de contribuição e mentoria de cada desenvolvedor interno em um determinado projeto, que é associado pelo próprio sistema a um novato interessado em contribuir com a comunidade.

As ferramentas descritas acima tem relação com o modelo de barreiras proposto por I. STEINMACHER, CHAVES *et al.* (2014), que também propõem como contribuição de seu trabalho, uma ferramenta para auxiliar novatos em software livre. Conhecida como FLOSSCoach⁷, tal ferramenta é uma plataforma de auxílio a novos contribuidores, que visa orientar novatos entre os principais caminhos que devem ser seguidos para realização de uma contribuição em projetos de software livre. Tais caminhos foram definidos a partir das principais dificuldades que novatos enfrentam neste contexto.

Além de trabalhos na literatura, uma diversidade de ferramentas também são propostas por desenvolvedores de fora do contexto acadêmico. O Up For Grabs⁸, por exemplo, é um website que tem como objetivo auxiliar novatos a encontrarem tarefas disponíveis em projetos de software livre. Os novatos acessam o website que apresenta a eles, uma série de projetos com tarefas nos sistemas de caça tarefas (do inglês, *issue tracker*) específicas para novos contribuidores, como mostra a Figura 2.3. Outro exemplo de projeto é o First Timers Only⁹, um website que estabelece diretrizes para novatos que desejam contribuir com projetos de software livre. Nesta página é possível encontrar uma diversidade de dicas e tutoriais relacionados aos primeiros passos de um novato em projetos neste contexto.

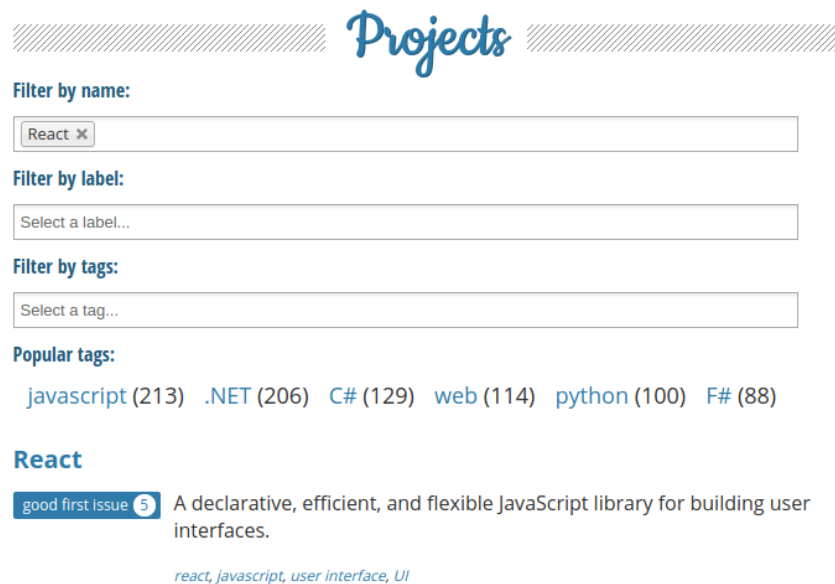


Figura 2.3: Exemplo de projeto cadastrado no Up For Grabs. React é o nome do projeto, e good-first-issue é o rótulo atribuído as tarefas destinadas a novatos.

⁷flosscoach.com

⁸up-for-grabs.net

⁹firsttimersonly.com

2.4.2 Suporte a documentação de software

Uma variedade de trabalhos relacionados também são propostos quanto a solução para problemas de documentação tais como os apresentados na Seção 2.3. [ZHONG e SU \(2013\)](#) propõem um mecanismo de identificação de erros em documentações de API's. Utilizando uma combinação de técnicas de processamento de linguagem natural e de análise de códigos fonte, os autores apresentam uma ferramenta para identificar trechos de documentação incoerentes e obsoletos. A ferramenta proposta, denominada DocRef, também identifica erros gramaticais. Tal ferramenta foi submetida a análise de cinco API's popularmente conhecidas e, de acordo com os autores, já reportou a mantenedores das cinco bibliotecas mais de mil erros de documentação.

Focados no processo de manutenção de programas escritos em Java, [HAIDUC *et al.* \(2010\)](#) propõem uma ferramenta para gerar sumários de classes e métodos em códigos fonte. Feita através de técnicas de sumarização de texto, a ferramenta visa facilitar o entendimento do código fonte de projetos Java, gerando um conjunto de palavras que descrevam corretamente as características principais de cada trecho de código. De acordo com os autores, quando desenvolvedores necessitam dar manutenção a uma determinada funcionalidade de um programa, dicas textuais podem auxiliar estes a compreenderem quais parte do código eles devem investigar, sem necessariamente gastarem um longo tempo compreendendo cada parte do projeto.

Em um estudo similar, [LI *et al.* \(2016\)](#) desenvolvem uma abordagem que gera automaticamente documentação para testes de unidade. Considerando a importância deste tipo de teste para manutenção e prevenção de erros em programas de computador, os autores desenvolveram uma ferramenta que analisa, a partir de um conjunto de técnicas, testes de unidade em repositórios de código fonte e preenche modelos pré-definidos de documentação com informações relevantes sobre cada teste de unidade. Tanto neste como no estudo de [HAIDUC *et al.* \(2010\)](#), questionários foram aplicados a desenvolvedores, a fim de avaliar a qualidade das documentações produzidas. Ambas as ferramentas tiveram grande aceitação por parte dos desenvolvedores, que compreendem o impacto da documentação de códigos na manutenção de projetos de software.

No contexto de software livre, [PANICHELLA \(2015\)](#) apresenta dois sistemas de recomendação que visam dar suporte a novatos em software livre. O primeiro, já mencionado anteriormente, é a ferramenta YODA, que recomenda mentores a novatos por meio da análise de dados dos projetos, como listas de e-mail e históricos de contribuição. E o segundo, nomeado CODES¹⁰, é um sistema de recomendação que sugere comentários para códigos Java, utilizando como fonte de dados, discussões entre desenvolvedores em meios de comunicação, tal como listas de e-mail e mecanismos de caça tarefa. Segundo os autores, sugerir mentores não é suficiente para dar suporte a novatos em software livre, auxiliá-los durante a compreensão e re-documentação dos projetos também são passos importantes na integração dos novatos a comunidades de software livre.

Fora do contexto acadêmico uma diversidade de projetos relacionados a este estudo dá suporte a documentações de software. Uma categoria comum são os geradores de

¹⁰spanichella.github.io/tools.html#codes

documentação. O Javadoc¹¹, por exemplo, analisa anotações e comentários em códigos Java, e gera automaticamente páginas HTML contendo informações sobre cada classe, método, interface e construtor desenvolvido. Como o Javadoc é específico para linguagem Java, é comum que outras linguagens apresentem outras alternativas, como o Pydoc¹², para linguagem Python, e o YARD¹³, para Ruby. É válido mencionar que, além dos geradores de documentação, outra categoria comum são as plataformas de hospedagem, tais como a Readthedocs¹⁴ e a Stoplight¹⁵, que fornecem modelos de documentação para que desenvolvedores organizem seus projetos online. A Figura 2.4 apresenta um exemplo de documentação gerada automaticamente pela ferramenta Javadoc.

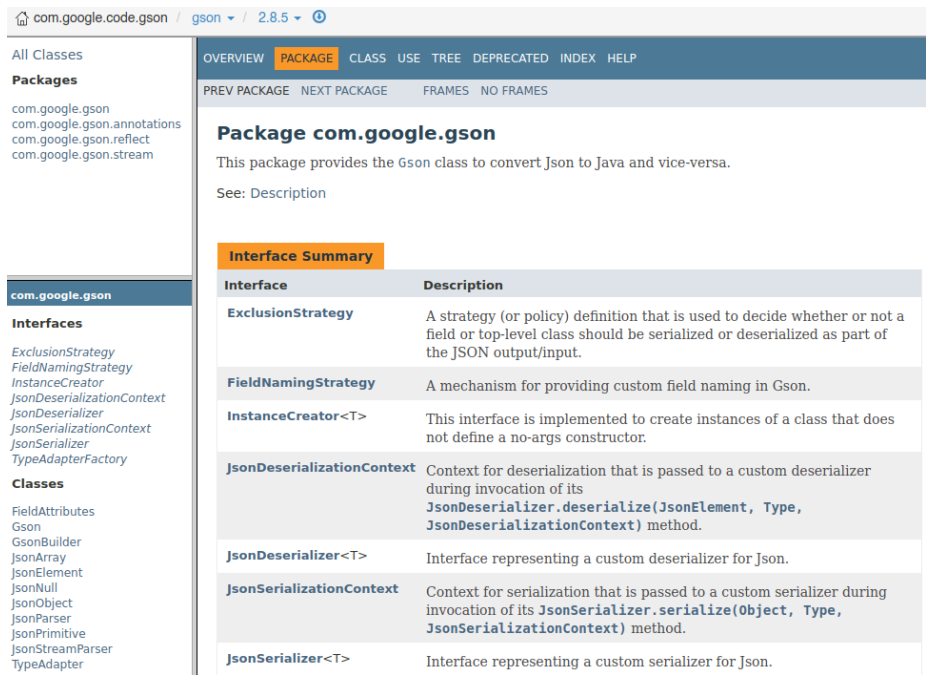


Figura 2.4: Exemplo de documentação gerada pela ferramenta Javadoc. Gson é o nome da classe contendo anotações e comentários de código.

¹¹docs.oracle.com/javase/10/javadoc
¹²docs.python.org/3/library/pydoc
¹³yardoc.org
¹⁴readthedocs.org
¹⁵stoplight.io

Capítulo 3

Metodologia

Levando em consideração as barreiras enfrentadas por novatos em software livre, em especial aquelas relacionadas à documentação dos projetos, propomos por meio desta pesquisa um modelo de classificação que contribua com a identificação de informações em documentações relevantes a novos contribuidores. A metodologia proposta dá continuidade a um trabalho preliminar (FRONCHETTI *et al.*, 2019), na qual foram investigadas características que influenciam a atratividade de novatos em projetos de software livre.

A elaboração deste método é baseada em dois estudos relacionados, um deles proposto por I. STEINMACHER, CHAVES *et al.* (2014), que evidenciaram a necessidade de uma ferramenta que auxilie novatos na compreensão de documentações de projetos em de software livre e outro proposto por PANICHELLA *et al.* (2015), que desenvolveram um método de identificação de seções em documentações de software, similar ao apresentado para este estudo.

3.1 Método

Para que seja possível responder as questões de pesquisa apresentadas na Seção 1.2, o método proposto é dividido em um conjunto de cinco etapas. Primeiro, são definidas categorias de informação a serem identificadas em arquivos de contribuição de projetos de software livre. Em seguida, documentações de um conjunto de projetos são extraídas como amostra, e uma análise qualitativa é executada para identificar trechos das documentações que sejam relevantes a novos contribuidores. Por fim, o modelo de classificação é desenvolvido e avaliado. A Figura 3.1 apresenta a sequência de atividades executadas no método desta pesquisa, e as seções a seguir descrevem detalhadamente cada uma das etapas executadas.

3.1.1 Definição das categorias

Um conjunto de categorias de informação foram elaboradas para facilitar a identificação de trechos relevantes a novos contribuidores nas documentações de projetos de software livre. Estas categorias foram estabelecidas com base nas barreiras apresentadas pelo modelo de I. STEINMACHER, CONTE *et al.* (2016) e em demais trabalhos relacionados,

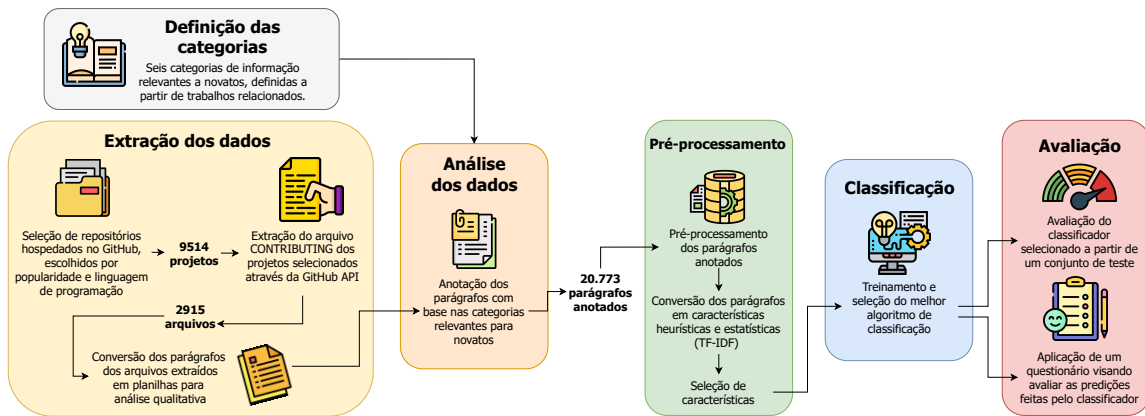


Figura 3.1: Método de pesquisa.

que também identificaram dificuldades que novatos enfrentam neste contexto (KROGH *et al.*, 2003; HANNEBAUER e GRUHN, 2017). Os trabalhos relacionados foram discutidos pelos pesquisadores em reuniões, visando encontrar categorias que estivessem de acordo com as dificuldades que novatos enfrentam. Na Tabela 3.1, apresentamos o conjunto contendo seis categorias que identificamos nas documentações de projetos de software livre. As principais dificuldades enfatizadas nas categorias estabelecidas foram aquelas passíveis de solução por meio da documentação dos projetos. Cada categoria representa a descrição de uma barreira retirada do modelo apresentado por I. STEINMACHER, CONTE *et al.* (2016) e das informações disponíveis na plataforma de suporte a novatos FLOSSCOACH (Igor STEINMACHER *et al.*, 2018), também apresentada pelos autores neste estudo. Estas categorias devem contribuir com o processo de contribuição dos novatos, já que abrangem problemas importantes, como a compreensão do projeto, contato com a comunidade e submissão de mudanças ao código fonte.

3.1.2 Extração dos dados

Seleção dos repositórios

Considerando que a premissa principal deste estudo é conseguir identificar trechos em documentações de projetos de software livre, um conjunto de projetos hospedados na plataforma de codificação GitHub foi selecionado como amostra. Visando maximizar a pluralidade dos projetos escolhidos, e buscando escapar de repositórios vazios ou sem atividade na plataforma, a seleção de projetos foi feita a partir da popularidade e linguagem de programação de cada repositório. Para compor nossa amostra, os projetos mais populares com predominância nas seguintes linguagens de programação foram escolhidos: JavaScript, Python, Java, PHP, C#, C++, TypeScript, Shell, C, e Ruby. A escolha de tais linguagens de programação se baseia em dados do evento GitHub Octoverse (GITHUB, 2020), que lista anualmente as linguagens de programação mais populares na plataforma. Já a filtragem por popularidade se baseou no número de estrelas de cada repositório no GitHub que, de acordo com BORGES *et al.* (2016), pode servir como métrica para definir quão popular os projetos são. Um total de 9.514 repositórios foram selecionados para compor nossa amostra por meio da GitHub API, um serviço que fornece dados sobre repositórios de código hospedados nesta plataforma (GITHUB, 2022).

Categoria	Descrição
Compreender o fluxo de contribuição (CF)	Não é incomum que os recém-chegados se sintam perdidos ou desmotivados quando não está claro como contribuir com um projeto de software livre. Para esta categoria, identificamos trechos de documentação que descrevem qual é o fluxo de contribuição de um projeto. O fluxo de contribuição pode ser definido como um conjunto de etapas que um recém-chegado precisa seguir para desenvolver uma contribuição aceitável para o projeto.
Escolher uma tarefa (ET)	Muitos desenvolvedores estão interessados em contribuir com projetos de software livre, mas a maioria deles não sabe com qual tarefa começar. Nesta categoria, identificamos frases descrevendo como os recém-chegados podem encontrar uma tarefa para contribuir com o projeto.
Contactar a comunidade (CC)	Além de um mentor, também é importante que os recém-chegados entrem em contato com a própria comunidade do projeto. Por esse motivo, identificamos nesta categoria qualquer informação que detalhasse como um recém-chegado pode entrar em contato com os membros da comunidade, incluindo links para canais de comunicação, tutoriais sobre como enviar uma mensagem, etiquetas de comunicação, entre outros.
Construir o ambiente de trabalho (CA)	Recém-chegados relataram em estudos anteriores que não encontraram explicações sobre como poderiam construir seu próprio ambiente de trabalho (construir, compilar, executar, gerenciar dependências, etc.) antes de contribuir. Para esta categoria, identificamos frases que explicassem como um novato poderia construir seu ambiente local de trabalho.
Lidar com código (LC)	Muitos projetos têm seus próprios padrões de código, arquiteturas e práticas de software. Nesta categoria, identificamos frases na documentação dos projetos descrevendo como o código deveria ser escrito, organizado e documentado pelos novos contribuidores.
Submeter mudanças (SM)	O último passo no processo de contribuição é a submissão de mudanças ao repositório do projeto. Nesta categoria, identificamos informações sobre como a submissão de alteração (em inglês, patch) deveria ser feita pelos novatos.

Tabela 3.1: *Categorias estabelecidas para identificação nas documentações dos projetos.*

Extração dos arquivos de contribuição

De cada um dos projetos selecionados, foi extraído o arquivo CONTRIBUTING do repositório de código fonte. A razão para escolha deste arquivo e não de outras fontes de documentação, se concentra no fato deste apresentar informações relevantes a novos contribuidores, ser de fácil extração e ser usualmente escrito a partir de uma mesma sintaxe, o que facilitaria o processo de identificação das categorias. Visto que os projetos selecionados estavam hospedados na plataforma de codificação GitHub, a documentação dos projetos também foram extraídas por meio da GitHub API. Para garantir que os projetos a serem estudados continham um arquivo CONTRIBUTING válido para análise, foram definidos um conjunto de filtros de seleção. Projetos que não continham o arquivo CONTRIBUTING, não continham o arquivo escrito em inglês, não continham o arquivo em formato Markdown, ou que continham o arquivo com tamanho menor que 0.5kB, foram removidos da amostra.

Ao fim da filtragem, o arquivo CONTRIBUTING de 2.915 projetos de software livre foram escolhidos para análise qualitativa. As linguagens de programação com maior número de projetos válidos foram TypeScript (n = 469), JavaScript (n = 399) e Ruby (n = 328). O total de 6599 projetos foram removidos após a filtragem, com uma média de 660 repositórios removidos por linguagem, sendo Java (n = 221), C (n = 172) e Shell (n = 172) as linguagens com menor número de projetos inclusos na amostra final, e a ausência do arquivo CONTRIBUTING o principal motivo de exclusão (n = 6055). Os motivos pelos quais certos projetos foram removidos são apresentados na Tabela 3.2.

Removido porque o arquivo CONTRIBUTING:	JavaScript (n=824)	Python (n=929)	Java (n=942)	PHP (n=941)	C# (n=990)	C++ (n=942)	TypeScript (n=990)	Shell (n=990)	C (n=947)	Ruby (n=1019)
Estava faltando	381 (46%)	527 (57%)	692 (73%)	593 (63%)	651 (66%)	604 (64%)	474 (48%)	785 (79%)	702 (74%)	646 (63%)
Não estava em inglês	3 (> 1%)	12 (1%)	4 (> 1%)	4 (> 1%)	3 (> 1%)	2 (> 1%)	3 (> 1%)	4 (> 1%)	2 (> 1%)	1 (> 1%)
Não estava em Markdown	2 (> 1%)	91 (10%)	5 (1%)	4 (> 1%)	1 (> 1%)	14 (1%)	4 (> 1%)	6 (1%)	26 (3%)	11 (1%)
Tinha tamanho menor que 0.5kB	41 (5%)	37 (4%)	22 (2%)	49 (5%)	54 (5%)	35 (4%)	42 (4%)	23 (2%)	28 (3%)	33 (3%)
Total de projetos removidos	425 (52%)	661 (71%)	721 (76%)	647 (68%)	709 (71%)	651 (69%)	521 (52%)	818 (82%)	755 (79%)	691 (67%)

Tabela 3.2: Número de projetos removidos por linguagem e as razões para exclusão. O valor “n” representa o total de projetos extraídos para determinada linguagem. É válido mencionar que certos arquivos podem ter sido removidos por mais de uma razão.

Conversão dos arquivos em planilhas para análise

Com apenas arquivos CONTRIBUTING válidos na amostra, os arquivos Markdown extraídos dos repositórios de código foram convertidos para planilhas em Excel. As planilhas, cada qual representando um projeto, foram organizadas em sete colunas. Na primeira coluna, os parágrafos do arquivo CONTRIBUTING foram divididos entre as células da coluna. A divisão dos parágrafos foi implementada com base na definição fornecida pelo próprio GitHub ([GitHub, s.d.](#)), que define parágrafo como “uma sequência de linhas não brancas que não podem ser interpretadas como outras estruturas de bloco”. As colunas seguintes foram reservadas para identificação das categorias de informação listadas na Tabela 2.3, a serem marcadas durante a análise qualitativa. Uma coluna adicional as criadas para identificação das categorias também foi inserida a fim de marcar os parágrafos que

não continham nenhuma das categorias de informação relacionadas. Para essa coluna foi atribuído uma categoria adicional chamada de Nenhuma categoria identificada (NC).

3.1.3 Análise dos dados

Com os arquivos em formato de planilha, foi realizada a análise qualitativa dos parágrafos para os projetos considerados válidos. Dois pesquisadores ficaram responsáveis por analisar os dados, um sendo aluno de mestrado em Ciência da Computação, e outro doutor em Ciência da Computação. Para assegurar que ambos obtivessem um padrão similar na identificação das categorias de informação, um conjunto de trinta projetos foram escolhidos aleatoriamente para análise e discussão. A análise dos trinta projetos foi dividida em três etapas consecutivas, e as planilhas analisadas da seguinte forma: Com a planilha de um projeto aberta, o pesquisador lia os parágrafos na primeira coluna e, quando julgasse tal informação pertencente a uma categoria de informação, adicionava uma marcação a coluna da respectiva categoria na mesma linha do parágrafo. Em nosso método, ficou decidido que cada parágrafo poderia receber apenas uma única categoria, já que a quantidade de informação oferecida por parágrafo, de modo geral, seria suficiente apenas para uma e não múltiplas categorias. Os parágrafos sem categoria identificada ficariam marcados com a categoria Nenhuma categoria identificada (NC).

Ao final de cada etapa, os pesquisadores discutiam as planilhas analisadas e definiam acordos a serem seguidos durante a análise qualitativa. Ao fim da primeira etapa, os pesquisadores atingiram uma concordância de 47.81% a partir de 10 planilhas analisadas. Para as duas últimas etapas, as concordâncias chegaram a 74.78%, o que foi considerado pelos pesquisadores como um valor suficiente de concordância, dado a complexidade do problema em questão. Ao final da terceira e última etapa, ambos os pesquisadores passaram então a analisar um novo conjunto de 500 planilhas retiradas da amostra de projetos válidos. Nenhum destes projetos havia sido utilizado na análise preliminar. A anotação ocorreu da mesma forma, com no máximo uma categoria anotada por parágrafo.

3.1.4 Descrição dos dados analisados

Ao fim da análise qualitativa, os pesquisadores chegaram a um conjunto contendo 20.733 parágrafos analisados. A distribuição de parágrafos analisados por arquivo é apresentada na Figura 3.2. Destes parágrafos, 13.272 (64%) foram identificados como pertencentes a uma das seis categorias de informação relevantes a novatos, e os demais sem categoria identificada. Um comparativo da média de parágrafos identificados por categoria em cada arquivo de contribuição, e o número de projetos identificados como contendo cada categoria são apresentados na Figura 3.3. A categoria com maior número de parágrafos identificados para o conjunto de projetos analisados foi a de Submeter mudanças (SM), com um total de 6.174 parágrafos relacionados, e aparecendo em média em 12 parágrafos por arquivo de contribuição. A categoria Submeter mudanças (SM) não só apresentou uma média de parágrafos por arquivo maior que as demais categorias, como também apareceu em mais projetos, tendo sido identificada em 396 (79%) dos 500 projetos analisados.

Seguindo a média de parágrafos identificados por arquivo, se encontraram em segundo e terceiro lugar as categorias Lidar com código (LC) e Compreender o fluxo de contribuição

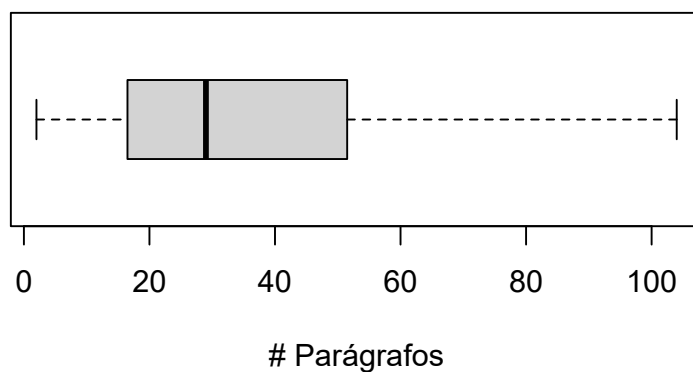


Figura 3.2: Número de parágrafos analisados por arquivo.

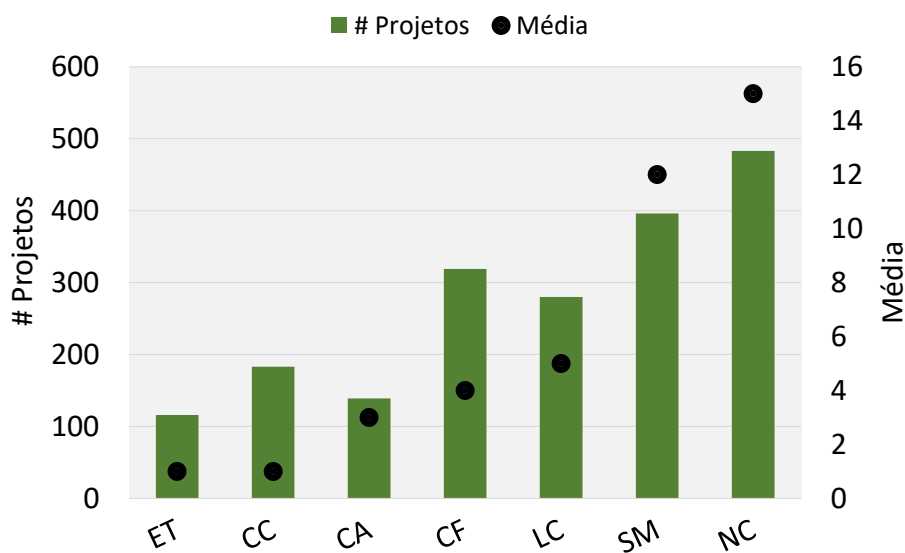


Figura 3.3: Número de projetos contendo cada categoria de informação, e média de parágrafos em determinada categoria por projeto.

Legenda: ET (Escolher a tarefa), CC (Contactar a comunidade), CA (Construir o ambiente de trabalho), CF (Compreender o fluxo de contribuição), LC (Lidar com o código), SM (Submeter mudanças), NC (Nenhuma categoria identificada).

(CF), respectivamente. Embora a categoria Lidar com código (LC) tenha aparecido em uma média de 5 parágrafos por arquivo analisado, enquanto que a categoria Compreender o fluxo de contribuição (CF) apareceu em média em 4 parágrafos, a categoria Compreender o fluxo de contribuição (CF) apareceu em mais projetos. Dos 500 projetos analisados, a categoria Compreender o fluxo de contribuição (CF) apareceu em 319 projetos (63%), um número relativamente maior ao da categoria Lidar com código (LC), que apareceu em apenas 280 projetos (56%).

Nas últimas posições se encontram as categorias Construir o ambiente de trabalho (CA), com uma média de 3 parágrafos por arquivo de contribuição e aparecendo em 139 projetos (27%), seguida por Contactar a comunidade (CC) e Escolher uma tarefa (ET), ambas aparecendo em média em 1 parágrafo por projeto, e tendo sido identificadas em 183 (36%) e 116 (23%) projetos, respectivamente. Além da média de parágrafos por categoria, também é apresentado na Figura 3.3 o número médio de parágrafos identificados como sem categoria de informação, ao qual a categoria Nenhuma categoria identificada (NC) é atribuída. Uma média de 14 parágrafos não tiveram nenhuma categoria identificada por arquivo de contribuição, tendo 483 projetos (96%) apresentado parágrafos sem nenhuma categoria, com a maioria dos projetos apresentando.

Outra informação interessante é encontrada no número de categorias de informação atendidas por projeto. Como mostra a Figura 3.4, todos os projetos analisados tiveram ao menos uma das categorias de informação identificada em seus parágrafos. Apenas 50 projetos (10%) apresentaram cinco ou mais categorias de informação, tendo 12 (2%) destes apresentado as seis categorias em seus arquivos de contribuição. Em contrapartida, 74% dos projetos apresentaram arquivos contendo entre duas e quatro categorias de informação. Um total de 135 projetos (27%) apresentaram três categorias, 130 projetos (26%) duas categorias, e 107 projetos (21%) quatro categorias. Por fim, 78 projetos (15%) apresentaram apenas uma categoria em seus arquivos de contribuição.

3.1.5 Pré-processamento

Preparação dos parágrafos

A preparação dos dados para classificação, definida pelas etapas de pré-processamento e extração de características na Figura 3.5, consistiu em preparar e dividir o conteúdo das planilhas analisadas entre características e rótulos para implementação do classificador. Neste estudo, consideramos como características os parágrafos das planilhas analisadas, e rótulos a categoria identificada pelos pesquisadores para cada um dos parágrafos. Antes de transformarmos os parágrafos em características para classificação, uma etapa de pré-processamento destes dados foi executada. Nesta etapa, informações não relevantes à classificação foram removidas a partir da combinação de três métodos comuns ao processamento de linguagem natural: lematização (THANAKI, 2017), remoção de pontuações (LANTZ, 2013), e remoção de palavras irrelevantes (Em inglês, *stopwords*).

No processo de lematização, variações de uma mesma palavra foram agrupadas em sua forma raiz a fim de facilitar o entendimento do conteúdo dos parágrafos por parte do classificador (e.g., em inglês, as palavras “*studies*” e “*studying*” foram reduzidas à sua forma raiz “*study*”). A remoção de pontuações foi feita através de expressões regulares e

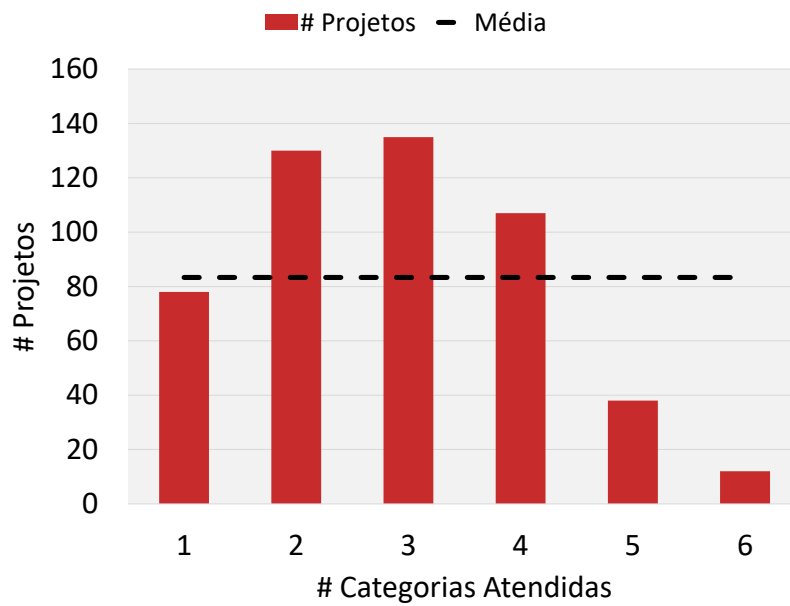


Figura 3.4: Número de categorias de informação atendidas por projeto, e linha média de 83 projetos por número de categorias atendidas.

métodos de substituição de palavras, e a remoção de palavras irrelevantes feita a partir de um conjunto de palavras pré-definido, removidas por meio de código. Neste estudo, o processo de lematização e remoção de palavras irrelevantes foi feito através da biblioteca NLTK¹, amplamente utilizada em estudos envolvendo processamento de linguagem natural (BONACCORSO, 2017b; LOPER e BIRD, 2002; BIRD, 2006).

Conversão dos parágrafos em características

Para descrever o conteúdo dos projetos analisados em informações válidas para classificação, os parágrafos das 500 planilhas analisadas foram convertidos em características estatísticas e heurísticas. Foram considerados como valores estatísticos, a transformação dos parágrafos em características TF-IDF² obtidas com o uso de bibliotecas de aprendizado de máquina. Os valores heurísticos foram obtidos por meio da análise manual dos parágrafos e categorias marcadas. Quando um padrão linguístico que associava parágrafos a uma categoria era identificado pelos pesquisadores, um novo valor heurístico era então gerado como característica para classificação.

A ideia de valores heurísticos emergiu do estudo realizado por PRANA *et al.* (2019) que identificou, por exemplo, que a categoria “cabecalho da documentação” proposta em seu estudo, poderia ser identificada pela ocorrência do título do repositório de código na documentação analisada. Além da divisão entre rótulos e características para classificação, os valores convertidos também foram divididos em conjuntos de treino e teste, a serem

¹<https://www.nltk.org/>

²TF-IDF (do inglês, *Term Frequency - Inverse Document Frequency*) é uma técnica estatística de extração de características textuais, que tem como principal objetivo identificar palavras relevantes em um texto não ou semi estruturado (KIDO *et al.*, 2014)

utilizados nas etapas seguintes.

Categoria	Características
Compreender o fluxo de contribuição (CF)	Clone, push, merge, pull request, contribution
Escolher uma tarefa (ET)	Issue, issue tracker, label, fork
Contactar a comunidade (CC)	Mailing list, contact, email, conduct, slack
Construir o ambiente de trabalho (CA)	Tool, package, update, dependencies, source code, setup
Lidar com código (LC)	Code snippet, library, debug, coding convention, method, variable
Submeter mudanças (SM)	Commit, diff, review, test, fetch, continuous integration

Tabela 3.3: *Exemplos de características heurísticas utilizadas.*

Seleção de características

Visando aprimorar o processo de classificação dos parágrafos, uma etapa de seleção de características foi executada. Nesta etapa, a função `SelectPercentile`³ foi utilizada para selecionar apenas as melhores características para treinamento. Tal função atribui pontuações para o conjunto de características disponível para treinamento através de testes estatísticos, e é capaz de selecionar apenas as melhores características a partir de um percentil definido como entrada pelo usuário. Nesta pesquisa, 15% das características geradas na etapa anterior foram mantidas, sendo estas as com melhores pontuações obtidas através da função `SelectPercentile`. O teste estatístico escolhido para esta seleção foi o Qui-quadrado, comum a problemas de classificação (BONACCORSO, 2017d; BROWNLEE, 2019).

3.1.6 Classificação

Para que fosse possível automatizar o processo de identificação das categorias em documentações de projetos de software livre, um modelo de classificação de categorias foi implementado. A elaboração deste modelo aconteceu em três etapas: Primeiro, cinco algoritmos de aprendizagem supervisionada foram combinados com estratégias de classificação multi-classe para treinar diferentes classificadores utilizando as características disponíveis para treinamento. Em seguida, um processo de avaliação da performance destes modelos foi executado, de modo a encontrar aquele que melhor identificasse as categorias em documentações de projetos de software livre. Tal modelo foi então treinado novamente com todas as características disponíveis para classificação, e avaliado através de uma análise quantitativa e um questionário com programadores.

³https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html

A implementação destes algoritmos, bem como todo processo de classificação, foi feita com auxílio da biblioteca scikit-learn (PEDREGOSA *et al.*, 2011), comumente utilizada em estudos envolvendo aprendizagem supervisionada. Uma ilustração completa do processo de classificação é apresentada na Figura 3.5.

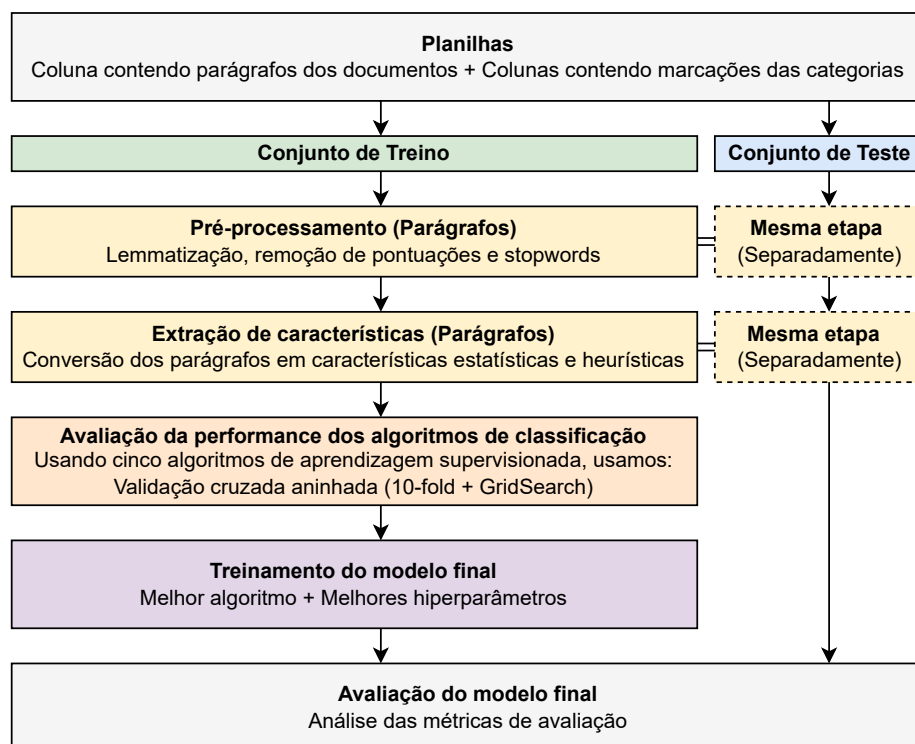


Figura 3.5: Construção do modelo de classificação.

Algoritmos de classificação

Diferentes classificadores foram treinados a fim de se obter o modelo de classificação que melhor identificasse trechos relevantes a novatos em documentações de projetos em software livre. Entre os algoritmos utilizados para treinamento dos classificadores, estavam: Random Forest (RF), Linear Support Vector Machine (SVM), Multinomial Naive Bayes (NB), K-Neighbors (kNN) e Logistic Regression (LR). Estes algoritmos foram escolhidos por serem amplamente utilizados em aprendizagem de máquina (BONACCORSO, 2017a), além de terem sido utilizados em trabalhos relacionados, como nos estudos de PRANA *et al.* (2019) e PANICHELLA *et al.* (2015). A busca pelos melhores hiper-parâmetros de cada algoritmo foi feita por meio de uma busca exaustiva através do método GridSearch (KLOSTERMAN, 2019), utilizando de pequenas variações dos valores padrão da biblioteca scikit-learn, especificados manualmente em código.

Para comparar a performance dos algoritmos selecionados com estratégias mais simples de predição, dois algoritmos de classificação aleatórios também foram utilizados como base na avaliação. Um dos algoritmos retornando sempre a classe mais frequente entre as instâncias no conjunto de treinamento, e o outro atribuindo classes de maneira completamente aleatória. A ideia por trás destes algoritmos aleatórios era avaliar se os modelos treinados a partir dos algoritmos de classificação selecionados realmente conseguem identificar as

classes, ou se estão apenas atribuindo categorias aos parágrafos de maneira equivocada. Tal abordagem já foi anteriormente utilizada por outros trabalhos relacionados (OWHADI-KARESHK *et al.*, 2019; SANTOS *et al.*, 2021).

Adaptações para classificação multi-classe

Como foi estabelecido que seis categorias de informação poderiam ser identificadas nas documentações dos projetos, duas estratégias de classificação multi-classe foram utilizadas em conjunto com os algoritmos selecionados, sendo elas One-vs-Rest e One-vs-One (BONACCORSO, 2017c). Na estratégia One-vs-Rest (OvR), classificadores binários foram treinados separadamente para identificar cada categoria de informação. Cada classificador ficou responsável por distinguir uma categoria das demais existentes, e uma função de decisão ficou responsável por atribuir categorias a novos parágrafos de entrada. Na estratégia One-vs-One (OvO), classificadores binários foram treinados para identificar as seis classes de informação em pares, cada par buscando diferenciar uma categoria de outra, e uma função de decisão também foi utilizada na categorização de novas entradas.

Por se tratar de um problema multi-classe com classes desbalanceadas, o uso da técnica de sobreamostragem SMOTE também foi avaliado nesta etapa. Nesta técnica, novas amostras são criadas para classes desfavorecidas, a fim de se aproximar o número de amostras disponíveis por classe (CAMPESATO, 2021). O objetivo de testar a técnica SMOTE, era de compreender se uma maior igualdade entre as amostras por classe afetaria ou não a performance dos algoritmos.

Escolha do melhor modelo de classificação

Na busca pelo modelo de classificação que melhor identificasse as categorias de informação relevantes a novatos em software livre, os algoritmos, técnicas e estratégias de classificação listados anteriormente foram submetidos a uma etapa de avaliação de performance. Nesta etapa, cada algoritmo de classificação foi treinado utilizando as duas técnicas de classificação multi-classe separadamente, bem como foram treinados com e sem a técnica de sobreamostragem SMOTE. O objetivo principal desta etapa era de quantificar a performance de diferentes combinações para classificação. Para melhor avaliar a capacidade de predição de cada uma das combinações, a técnica de validação cruzada aninhada k -fold e a métrica de avaliação f -measure foram utilizadas.

Na validação cruzada aninhada k -fold, dois laços de repetição aninhados, cada qual com k repetições, são utilizados para particionar o conjunto de treinamento com o objetivo de avaliar o comportamento de um único algoritmo de classificação sob diferentes subconjuntos de dados. Nesta pesquisa, consideramos $k = 10$ partições. Nesta estratégia, o laço de repetição externo divide o conjunto de treinamento em k partes iguais. Para cada iteração i deste laço, o subconjunto k_i de dados é guardado para avaliação da performance, e os $k - 1$ subconjuntos restantes usados para treinamento de um modelo de classificação. Em cada iteração do laço externo, o laço interno também divide os $k - 1$ subconjuntos para treinamento em k diferentes partes iguais, e os utiliza para computar a melhor combinação de hiper-parâmetros para aquele algoritmo através do método GridSearch.

O laço de repetição externo é executado para avaliar a capacidade de generalização⁴ dos algoritmos de classificação selecionados em relação ao problema definido nesta pesquisa, e o laço interno, para encontrar os melhores hiper-parâmetros para cada algoritmo. Após o processo de validação cruzada aninhada, as performances dos modelos gerados pelo laço externo são avaliadas pela métrica de avaliação *f-measure* utilizando o subconjunto de validação remanescente a cada iteração.

Sobre a métrica de avaliação *f-measure* nesta pesquisa, temos que ela é responsável por definir qual dos algoritmos de classificação selecionados obteve a melhor performance nos diferentes cenários gerados pela validação cruzada. A métrica *f-measure* para problemas de classificação multi-rótulo é definida da seguinte forma (PANICHELLA *et al.*, 2015):

$$F1 = \frac{\sum_{l \in L} w_l \times F1_l}{|L|} \quad (3.1)$$

$$F1_l = \frac{2 \times Precision_l \times Recall_l}{Precision_l + Recall_l}$$

onde w_l é a proporção do rótulo l em todos os dados previstos, $F1_l$ é o valor de *f-measure* para o rótulo l , L é o conjunto de rótulos, $Precision_l$ é a precisão para o rótulo l , e $Recall_l$ é o sensibilidade para o rótulo l . Ao computar precisão e sensibilidade para um determinado rótulo l , uma instância é considerada positiva se ela contém o rótulo l , caso contrário, é considerada negativa. Desta forma, temos que precisão é a proporção de instâncias positivas preditas que são realmente positivas, enquanto que sensibilidade é a proporção de instâncias positivas que foram preditas como positivas. Vale ressaltar que o valor de *f-measure* é calculado separadamente para os modelos gerados no laço externo da etapa de validação cruzada, e uma média aritmética destes valores é apresentada como resultado para o algoritmo de classificação testado neste processo. Consideramos como o algoritmo com melhor performance, aquele que obteve a melhor média aritmética entre os valores de *f-measure* obtidos na validação cruzada aninhada.

3.1.7 Avaliação do modelo de classificação

Assim que o modelo de classificação final foi definido, duas análises foram realizadas como parte resultante deste trabalho. Primeiro, uma avaliação da importância das características extraídas do modelo de classificação final foi executada. Em seguida, uma avaliação das previsões feitas pelo mesmo modelo foi executada através de um questionário com 46 desenvolvedores de software. A descrição de cada uma das atividades é definida detalhadamente abaixo.

⁴Generalização refere-se à capacidade de realizar previsões corretas em dados novos, anteriormente não vistos, em oposição aos dados usados para treinar o modelo de classificação (GOOGLE GLOSSARY, 2019)

Avaliação das características

Assim que um modelo de classificação final foi selecionado após a etapa de validação cruzada, uma investigação das características deste modelo foi executada. A ideia era compreender quais características tem mais importância na identificação de cada uma das categorias relevantes a novatos em software livre. Tanto as características estatísticas como as heurísticas foram avaliadas. Para que essa análise fosse possível, os pesos de cada uma das características foram extraídos do modelo de classificação elegido, através da própria biblioteca de aprendizado de máquina utilizada durante a construção do modelo. Quanto maior o peso de uma característica, consequentemente maior ficou considerada sua importância para identificação de cada uma das categorias. A relevância das principais características por categoria é apresentada por meio de gráficos de dispersão.

Além disso, uma avaliação da diferença de importância entre o conjunto de características estatísticas e heurísticas também foi efetuada. Para isso, dois novos modelos de classificação com as mesmas configurações do modelo final foram treinados, um contendo apenas o conjunto de valores estatísticos, e outro apenas o conjunto de valores heurísticos. O valor de *f-measure* foi calculado para ambos os modelos. A ideia era compreender qual conjunto de valores obteve maior impacto na identificação das categorias relevantes a novos contribuidores, promovendo uma discussão sobre os diferentes conjuntos de características utilizados.

Avaliação das previsões

Uma avaliação de previsões feitas pelo modelo de classificação final também foi executada. O objetivo desta etapa era compreender se novas previsões fariam sentido para contribuidores no contexto de desenvolvimento de software, e se as categorias preditas de fato teriam relação com o conteúdo dos parágrafos. Para isso, novos e experientes contribuidores em software livre foram convidados a participar de um questionário contendo previsões de categorias em parágrafos para avaliação. A seleção de participantes foi feita através da plataforma de *crowd sourcing* Amazon Mechanical Turk⁵, de onde apenas pessoas com experiência em programação foram convidadas para realizar o questionário.

A apresentação do questionário ficou dividida em três partes: treinamento, avaliação e coleta de dados demográficos. Na etapa de treinamento, a descrição das categorias preditas pelo modelo de classificação foram apresentadas aos participantes em formato similar a Tabela 3.1. Para confirmar que os participantes compreenderam a explicação dada, estes foram requisitados a responder uma série de seis perguntas, cada qual solicitando ao participante que selecionasse a definição correta pra cada categoria de informação. Responder a definição de cada categoria era pré-requisito para continuar o questionário, e os participantes que selecionaram a definição de duas ou mais categorias incorretamente foram excluídos da pesquisa.

Na etapa de avaliação, os participantes foram convidados a qualificar previsões feitas pelo classificador a parágrafos extraídos do arquivo de contribuição de 75 projetos de software livre. A escolha destes projetos foi feita de maneira aleatória, a partir da amostra de 2.915 repositórios coletada anteriormente. Nenhum dos 75 projetos selecionados foi

⁵<https://www.mturk.com/>

utilizado em etapas anteriores, sendo o conteúdo destes completamente desconhecidos pelo modelo de classificação. Para compor o conjunto de parágrafos a serem avaliados pelos participantes no questionário, categorias de informação foram atribuídas a cada um dos 75 projetos selecionados, totalizando um conjunto de 2.685 parágrafos classificados. Deste conjunto de parágrafos, 10 parágrafos por categoria de informação foram aleatoriamente selecionados para compor a amostra de questões do questionário.

Choose a task: Specifies the set of information that describes how new contributors can find a task to contribute with the project. It may also contain descriptions of different types of tasks designed for newcomers and guidelines on how to perform a new contribution.

Use the following scale to determine how adequate the category "Choose a task" relates to the paragraph below:

"Once the team have approved an issue/spec, development can proceed. If no developers are immediately available, the spec can be parked ready for a developer to get started. Parked specs' issues will be labeled "Help Wanted". To find a list of development opportunities waiting for developer involvement, visit the Issues and filter on the Help-Wanted label."

Extremely inadequate	Somewhat inadequate	Somewhat adequate	Extremely adequate
----------------------	---------------------	-------------------	--------------------

Figura 3.6: Exemplo de questão aplicada no questionário.

Durante a avaliação, cada participante ficou responsável por avaliar 3 predições feitas por categoria de informação em uma escala de avaliação Likert com quatro níveis, de extremamente inadequado a extremamente adequado. A escolha dos parágrafos a serem apresentados a cada participante foi feita de forma aleatória, resultando em um total de 18 parágrafos avaliados por pessoa. Após a etapa de avaliação, os participantes foram questionados sobre seus anos de experiência com programação e contribuição com comunidades de software livre.

Capítulo 4

Resultados da classificação

Neste capítulo, os resultados obtidos com o processo de elaboração de um modelo de classificação capaz de identificar categorias relevantes a novatos em software livre são discutidos. Primeiro, os resultados do processo de escolha do modelo de classificação são apresentados. Em seguida, a performance do classificador escolhido frente ao conjunto de teste é apresentada, e uma avaliação das importâncias das características deste mesmo classificador é discutida. Por fim, as avaliações feitas por desenvolvedores sobre as previsões do classificador escolhido são relatadas.

4.1 Escolha do modelo de classificação

Durante a escolha do modelo que melhor identificasse as categorias de informação relevantes a novatos, cinco algoritmos de classificação mais dois algoritmos de predição aleatória foram comparados. Cada algoritmo foi treinado em uma validação cruzada aninhada com duas estratégias de classificação multi-classe, bem como com e sem a técnica de sobre-amostragem SMOTE. Na Tabela 4.1, um comparativo dos valores de *f-measure* obtidos para cada combinação de algoritmo, estratégia de classificação e uso de sobre-amostragem são apresentados. Entre as performances obtidas, a combinação que obteve o maior valor de *f-measure* é a do algoritmo LinearSVC, treinado com a estratégia One-Vs-Rest (OvR), sem aplicação da técnica SMOTE, que obteve o valor de *f-measure* de 0.652 após a validação cruzada. De acordo com os resultados obtidos no laço interno da validação cruzada aninhada, o algoritmo LinearSVC teve sua performance máxima com hiper-parâmetros definidos para 1,000 iterações (max iter = 1000), regularização igual a 1 ($C = 1$), e tolerância igual a 0.001 (tol = 0.001).

Como definido anteriormente na metodologia desta pesquisa, esta combinação acabou por ser escolhida como aquela utilizada para treinar o modelo de classificação final desta pesquisa. Quanto a performance das demais combinações, temos que o mesmo algoritmo LinearSVC apresentou também a segunda melhor performance entre os algoritmos testados, com *f-measure* igual a 0.646, também sem o uso da técnica de sobre-amostragem SMOTE, só que treinado com a estratégia One-Vs-One (OvO). Por outro lado, o algoritmo com a pior performance para este problema foi o kNN, com valores de *f-measure* inferiores a 0.6 para todas as combinações testadas.

	Com SMOTE		Sem SMOTE	
	OvR	OvO	OvR	OvO
RF	0.636	0.625	0.620	0.609
kNN	0.563	0.566	0.516	0.530
SVC	0.630	0.634	0.652	0.646
LR	0.612	0.606	0.617	0.602
NB	0.579	0.580	0.636	0.633
Aleatório (Freq.)	0.001	0.009	0.001	0.190
Aleatório (Rand.)	0.001	0.010	0.001	0.010

Tabela 4.1: Valores de f -measure para os classificadores testados no processo de validação cruzada.

4.2 Avaliação do modelo de classificação

Com a melhor combinação de algoritmo, hiper-parâmetros, estratégia de classificação e uso de técnicas de sobre-amostragem definida, a mesma combinação foi treinada em um novo modelo de classificação, avaliado com base no conjunto de parágrafos para teste. Na Tabela 4.2, são apresentados os valores de f -measure, precisão e sensibilidade para cada uma das categorias de informação extraídas do modelo de classificação treinado. As categorias estão organizadas na tabela em ordem decrescente, com base no valor de f -measure obtido a partir do modelo. Ao final da tabela, a performance obtida com a identificação de parágrafos sem nenhuma categoria, bem como a performance geral deste novo modelo treinado são apresentadas. A performance geral segue o mesmo cálculo usado anteriormente na comparação dos diferentes modelos de classificação.

Categoria	F1	Precisão	Sensibilidade
Construir o ambiente de trabalho (CA)	0.716	0.674	0.764
Lidar com código (LC)	0.711	0.682	0.743
Contactar a comunidade (CC)	0.648	0.657	0.639
Submeter mudanças (SM)	0.617	0.717	0.541
Escolher a tarefa (ET)	0.379	0.687	0.261
Compreender o fluxo de contribuição (CF)	0.345	0.519	0.258
Nenhuma categoria identificada (NC)	0.592	0.596	0.588
Geral	0.651	0.655	0.662

Tabela 4.2: Performance do modelo de classificação final (LinearSVC).

De acordo com os valores obtidos, as categorias que melhor são identificadas pelo classificador são as de Construir o ambiente de trabalho (CA) e Lidar com código (LC), com f -measure igual a 0.716 e 0.711, respectivamente. Com valores próximos se encontram as categorias de Contactar a comunidade (CC) e Submeter mudanças (SM), apresentando, correspondentemente, os valores 0.648 e 0.617 para f -measure. Por fim, se encontram as ca-

categorias com as piores performances na classificação, Escolher a tarefa (ET) e Compreender o fluxo de contribuição (CF), com valores de f -measure de 0.379 e 0.345, respectivamente. Em relação a classe atribuída a parágrafos sem nenhuma categoria identificada, temos que o valor de f -measure é de 0.592.

Acredita-se que a baixa performance do classificador em relação a certas categorias de informação pode ser explicada por uma série de fatores. A categoria Escolher a tarefa (ET), por exemplo, é a categoria com a menor média de parágrafos identificados por arquivo analisado, aparecendo em média em 1 parágrafo por arquivo. A categoria Compreender o fluxo de contribuição (CF), por outro lado, apesar de apresentar uma média maior de parágrafos identificados por arquivo, tem uma definição que engloba informações mais gerais do projeto, o que pode ter gerado um fator de confusão para a generalização do classificador. Para promover uma visão mais clara dos erros durante a predição, é apresentado na Figura 4.1 a matriz de confusão do modelo de classificação final.

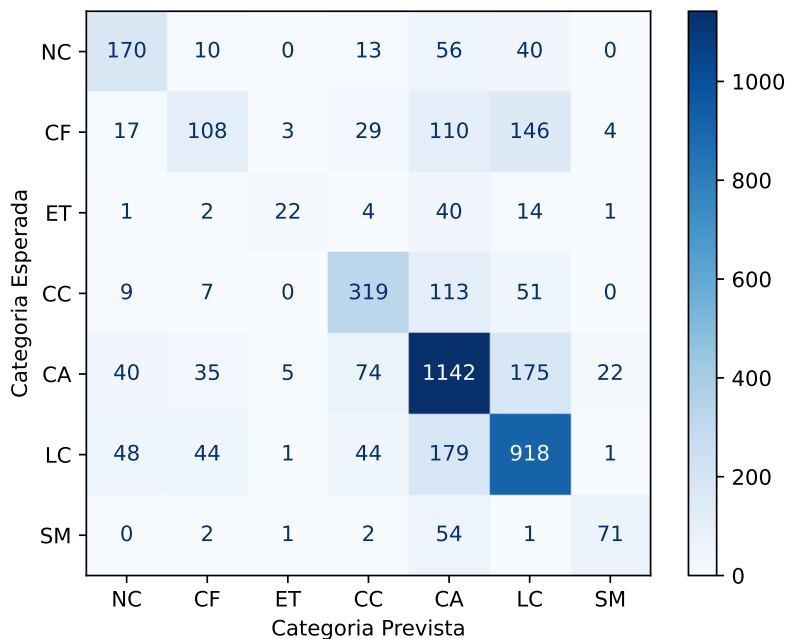


Figura 4.1: Matriz de confusão para o modelo de classificação final (LinearSVC).

Legenda: NC (Nenhuma categoria identificada), CF (Compreender o fluxo de contribuição), ET (Escolher a tarefa), CC (Contactar a comunidade), CA (Construir o ambiente de trabalho), LC (Lidar com o código), SM (Submeter mudanças).

A partir da matriz de confusão, é possível perceber alguns dos fatores de confusão discutidos. A categoria Compreender o fluxo de contribuição (CF), por exemplo, tem parte de suas predições atribuídas a outras categorias, como Construir o ambiente de trabalho (CA) e Lidar com código (CC), informações as quais podem aparecer também na descrição do fluxo de contribuição de um projeto. Outra observação interessante pode ser notada na relação entre as categorias Lidar com código (LC) e Construir o ambiente de trabalho (CA), ambas contendo erros de predição entre si. Essa confusão pode ser explicada pelo fato de que, por exemplo, ao lidar com o código de um projeto, comandos associados a construção

do ambiente de trabalho podem ser utilizados, tais como comandos em um terminal ou a instalação de um pacote.

4.3 Avaliação das características

Quanto a avaliação das características utilizadas no modelo de classificação final, duas análises foram feitas. Primeiro, apresentamos na Seção 4.3.1 uma comparação entre os conjuntos de características utilizados para treinar o modelo. Em seguida, apresentamos na Seção 4.3.2 uma avaliação das características mais importantes encontradas no modelo de classificação final.

4.3.1 Comparação entre conjuntos de características

Durante o treinamento do classificador, dois conjuntos de características foram utilizados, características estatísticas (TF-IDF) e características heurísticas, baseadas em um conjunto de regras definidos pela análise qualitativa dos parágrafos. Na Tabela 4.3, é apresentada uma comparação da performance desses dois conjuntos quando novos modelos utilizando as mesmas configurações do modelo final são utilizados para treinar apenas um dos conjuntos de dados.

Características	F1	Precisão	Sensibilidade
Estatísticas	0.651	0.657	0.664
Heurísticas	0.414	0.502	0.493

Tabela 4.3: Performance do modelo de classificação características específicas.

Segundo consta na tabela, o modelo treinado a partir do conjunto de características estatísticas apresentou o mesmo valor de *f-measure* encontrado no modelo de classificação final, 0.651. Em contrapartida, quando um novo modelo é treinado apenas com as características heurísticas, a performance apresentada é inferior ao modelo final, com um valor de *f-measure* igual a 0.414. Embora exista igualdade em performance entre o modelo com características estatísticas e o modelo final, foi decidido que nenhuma alteração seria feita no modelo de classificação final, visto que as características heurísticas não afetam negativamente a performance do mesmo.

4.3.2 Características com maior importância

Para melhor compreender quais características são importantes na identificação das categorias de informação relevantes a novatos em software livre, uma avaliação da importância das características utilizadas no modelo final foi executada. Como o modelo de classificação final foi treinado a partir do algoritmo LinearSVC, a importância das características foi definida através dos pesos atribuídos a cada característica pelo algoritmo de classificação. Quanto maior o peso de uma característica para uma categoria treinada pelo modelo, maior foi considerada sua importância na identificação da mesma categoria (CHANG e LIN, 2008).

Para garantir que os pesos extraídos de fato representassem as características com maior relevância para cada categoria de informação, o algoritmo LinearSVC foi executado com as mesmas configurações do modelo final em um processo de validação cruzada k-fold com 10 partições. Os pesos das características de cada um dos 10 modelos gerados pela validação cruzada foram utilizados para gerar uma média de peso por característica treinada. Na Figura 4.2, as características com as maiores médias de peso são apresentadas para cada categoria de informação identificada. Para ressaltar a diferença estatística entre os diferentes pesos por classe, o método Scott Knott Difference (ESD) foi utilizado nas cinco principais características de cada categoria. As cores na Figura 4.2 representam os diferentes grupos de efeito encontrados pelo método estatístico em questão (TANTITHAMTHAVORN *et al.*, 2018).

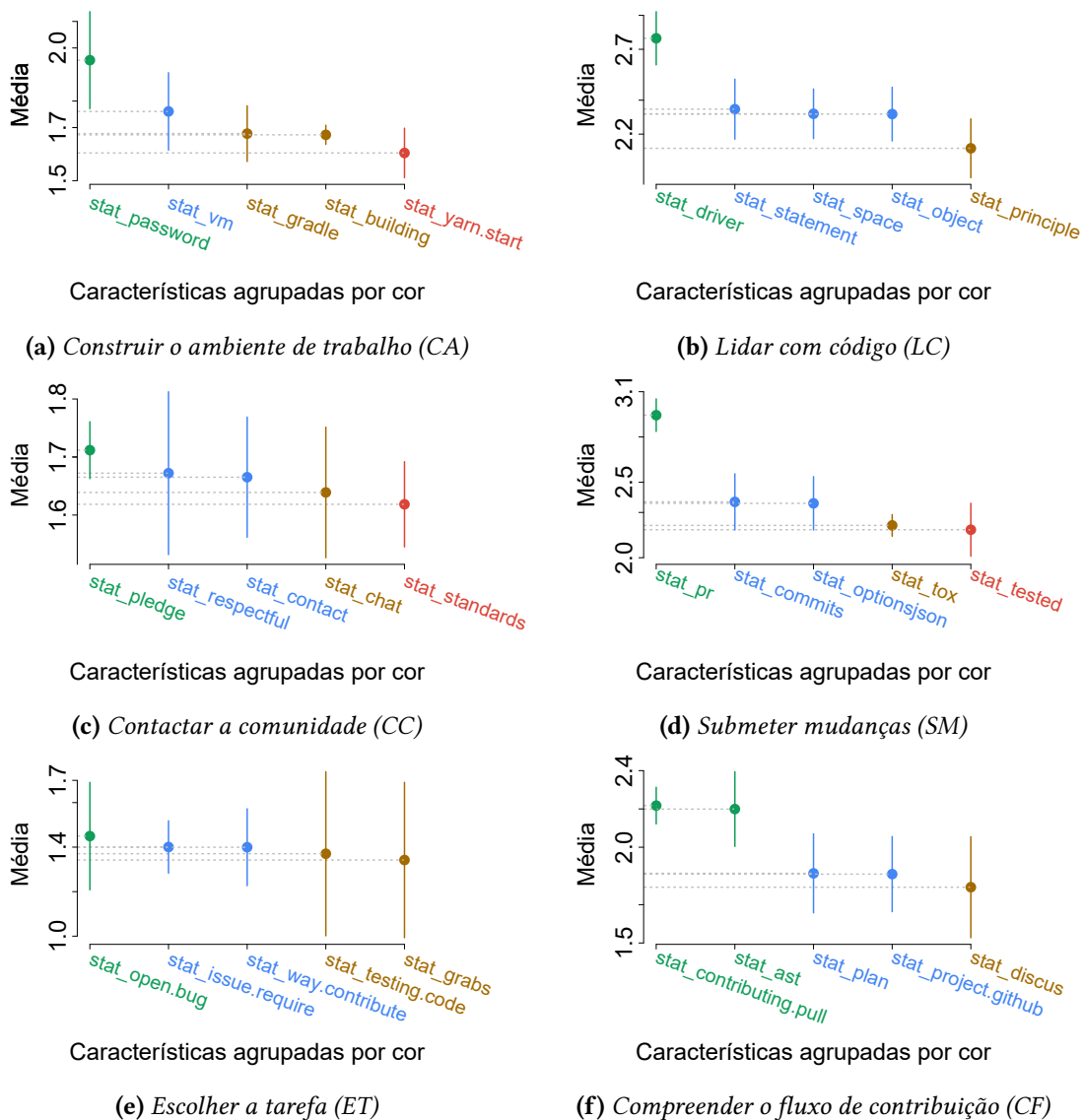


Figura 4.2: Características com maior importância para predição de cada categoria de informação.

A partir da Figura 4.2, é possível observar a relação entre cada categoria e as palavras com maior importância para sua predição. Para informações relacionadas a como Construir

o ambiente de trabalho (CA), por exemplo, palavras como *password* (do inglês, senha) e *vm* (sigla em inglês para máquina virtual) são consideradas importantes e possivelmente frequentes em parágrafos relacionados a tal categoria. Outra categoria interessante de se observar é a de Escolher uma tarefa (ET), com palavras como *bug* (do inglês, defeito em código) e *issue* (do inglês, tarefa) aparecendo entre as mais importantes. Observa-se a partir destas palavras que, de fato, as características com maior relevância para cada categoria fazem relação com suas descrições, corroborando com a premissa de que o classificador é capaz de identificar tais informações em texto.

4.4 Questionário sobre predições

Quanto ao questionário onde desenvolvedores foram convidados a avaliar as predições feitas pelo modelo de classificação final através da plataforma *Amazon Turk*, temos que a resposta de 46 participantes foram registradas. Destes 46 participantes, 29 foram removidos da pesquisa por não responderem as definições das categorias de informação corretamente ao início do questionário, ou por não assinalarem a opção correta na questão de validação de atenção. Ao remover os 29 participantes da pesquisa, um total de 17 respostas acabaram por serem consideradas válidas para análise. Embora o número de respostas válidas seja menor do que número total de respostas obtidas, entende-se que a avaliação das respostas obtidas em plataformas de *crowd-sourcing* é necessária, visto que o ato de remoção de respostas inválidas pode assegurar a qualidade e consistências dos dados obtidos, como apontam estudos na literatura (REID *et al.*, 2022; MAO *et al.*, 2017).

Na Tabela 4.4, são apresentados os dados demográficos coletados dos 17 participantes considerados como válidos para estudo. Como aponta a tabela, todos os respondentes tinham ao mínimo alguma experiência em programação, tendo 64% dos participantes de 3 a 15 anos de experiência na área. Em relação a experiência dos desenvolvedores com a manutenção de projetos de software livre, apenas 3 relataram jamais ter contato com projetos neste contexto, enquanto que os 82% restantes apontaram ter até 15 anos de experiência como mantenedores. Todos os 14 participantes com alguma experiência em manutenção de projetos de software livre apontaram já terem contribuído com a documentação dos projetos.

Experiência	Em programação	Em OSS
Nenhuma experiência	0	3
No máximo 3 anos de experiência	4	7
Mais de 3 anos e menos de 15 anos	11	7
Pelo menos 15 anos de experiência	2	0

Tabela 4.4: Experiência em programação e em software livre dos participantes do questionário.

Em relação as predições avaliadas pelos participantes, ao menos 30% das categorias atribuídas aos parágrafos pelo classificador foram consideradas muito adequadas, e 69% das predições avaliadas foram consideradas ao menos ligeiramente adequadas para categoria predita, como mostra a Figura 4.3. Em contrapartida, segundo a avaliação feita pelos participantes, em média 18% dos parágrafos receberam categorias de informação

ligeiramente inadequadas pelo classificador, e em média 9% dos parágrafos receberam uma categoria muito inadequada. De maneira geral, os resultados apontam uma maior incidência de categorias preditas corretamente pelo classificador, embora alguns parágrafos ainda recebam categorias consideradas inadequadas pelos participantes.

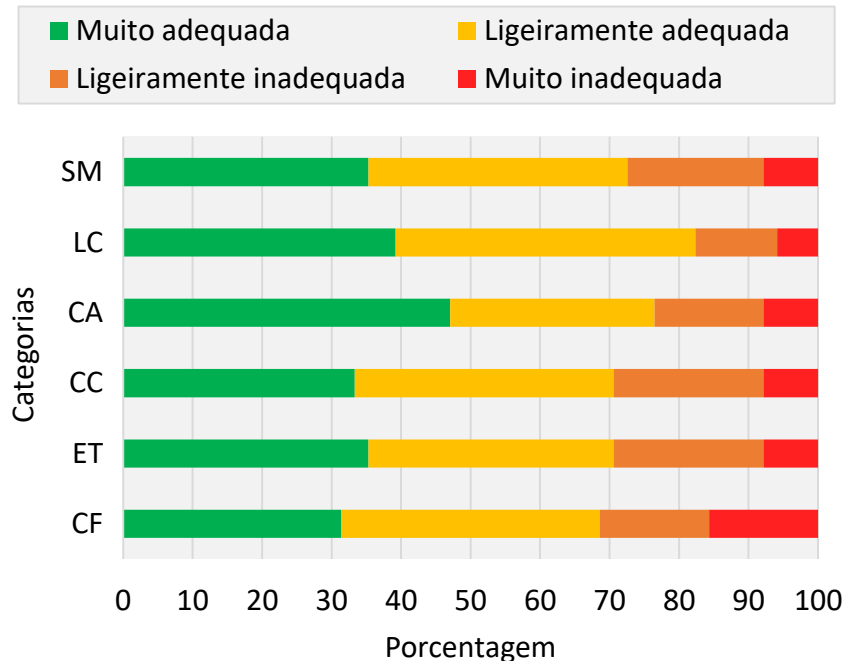


Figura 4.3: Avaliação das previsões feitas pelo modelo de classificação final, de acordo com participantes do questionário.

Legenda: CF (Compreender o fluxo de contribuição), ET (Escolher a tarefa), CC (Contactar a comunidade), CA (Construir o ambiente de trabalho), LC (Lidar com o código), SM (Submeter mudanças).

Quanto as avaliações individuais de cada categoria, temos que os parágrafos preditos com a categoria Construir o ambiente de trabalho (CA) foram os que obtiveram as melhores avaliações pelos participantes, sendo 47% destes parágrafos considerados muito adequados a categoria predita, e 76% destes ao menos ligeiramente adequados. Em segundo lugar, a categoria Lidar com o código (LC) obteve as melhores avaliações, com 39% dos parágrafos avaliados tendo a categoria considerada como muito adequada, e 82% ao menos ligeiramente adequados, 6% a mais do que a categoria Construir o ambiente de trabalho (CA). Todas as demais categorias seguiram um padrão similar de adequação, com 69% à 73% de seus parágrafos considerados pelo menos ligeiramente adequados as categorias preditas pelo classificador.

Em relação as categorias com os maiores níveis de inadequação, Compreender o fluxo de contribuição (CF) é a categoria que lideram as avaliações negativas, tendo 16% de suas previsões consideradas muito inadequadas aos parágrafos avaliados. Todas as demais categorias obtiveram menos da metade das avaliações negativas atribuídas a categoria Compreender o fluxo de contribuição (CF), com no máximo 8% de suas previsões consideradas muito inadequadas. Tanto as avaliações positivas quanto negativas atribuídas as categorias de informação parecerem corroborar com a performance apresentada na

Tabela 4.2, que apresenta as categorias Construir o ambiente de trabalho (CA) e Lidar com o código (LC) com os melhores valores de *f-measure*, enquanto que a categoria Compreender o fluxo de contribuição (CF) apresenta o pior valor entre todas as categorias.

4.5 Avaliação das documentações restantes

Nesta última etapa, os 2.271 arquivos de contribuição coletados que não foram utilizados na análise qualitativa foram submetidos ao modelo de classificação final para predição. O objetivo desta etapa era de avaliar a existência das categorias de informação nos parágrafos destes projetos, a fim de comparar os resultados obtidos com o que havida sido observado anteriormente na análise qualitativa (Seção 3.1.4). Na Figura 4.4, é apresentado o número de projetos com parágrafos preditos contendo cada categoria de informação, bem como o número médio de parágrafos em determinada categoria por projeto.

É possível observar que o número de projetos por categoria e o número médio de parágrafos por arquivo seguem padrão similar ao obtido na análise qualitativa (Figura 3.3), com a grande maioria dos projetos discutindo principalmente fases finais do processo de contribuição, incluindo a categoria sobre Como Submeter Mudanças (SM), que aparece em 2.192 arquivos analisados (96%) e tem uma média de 11 parágrafos por projeto, e a categoria Lidar com código (LC), que aparece em 1.661 projetos (73%) e tem uma média de 4 parágrafos por arquivo. Informações relacionadas as etapas iniciais do processo de contribuição, tais como Escolher a tarefa (ET) e Contactar a Comunidade (CC), aparecem de forma menos recorrente, ambas com uma média de menos de um parágrafo por arquivo avaliado, seguindo o mesmo padrão observado na análise qualitativa. O número de projetos em que ambas as categorias são observadas também é menor dos que as demais categorias, tendo a categoria Escolher a tarefa (ET) aparecido em 332 arquivos (14%), e a categoria Contactar a Comunidade (CC) em 530 (23%).

Mais uma vez ao centro do gráfico, as categorias sobre como Compreender o fluxo de contribuição (CF) e Construir o ambiente de trabalho (CA) apareceram ambas com uma média de 2 parágrafos por arquivo, tendo a primeira aparecido em 1.649 arquivos de contribuição (72%), e a segunda em 1.163 projetos (51%). Por fim, é válido ressaltar que o número de parágrafos sem categoria identificada é o maior entre todas as classes preditas pelo modelo de classificação, tendo a categoria Nenhuma categoria identificada (NC) aparecido em 2.265 projetos (99%), com uma média de 15 parágrafos por arquivo de contribuição. Embora algumas diferenças em porcentagem e número médio de parágrafos possam ser observadas em relação aos resultados obtidos na análise qualitativa, a proporção de instâncias por categoria é similar para ambos os conjuntos analisados, o que sugere que as proporções observadas podem representar a maneira como arquivos de contribuição são organizados.

Assim como feito para a análise qualitativa (Figura 3.4), a Figura 4.5 apresenta o número de categorias atendidas por projeto predito. Os números registrados para o conjunto de projetos preditos seguem padrão similar aos observados na análise qualitativa, com a maioria dos projetos (77%) apresentando entre 2 e 4 categorias de informação. Dos 2.271 arquivos analisados pelo classificador, 729 apresentaram quatro categorias (32%), 602 três categorias (27%), e 410 duas categorias (18%), sendo todos estes valores acima da média de

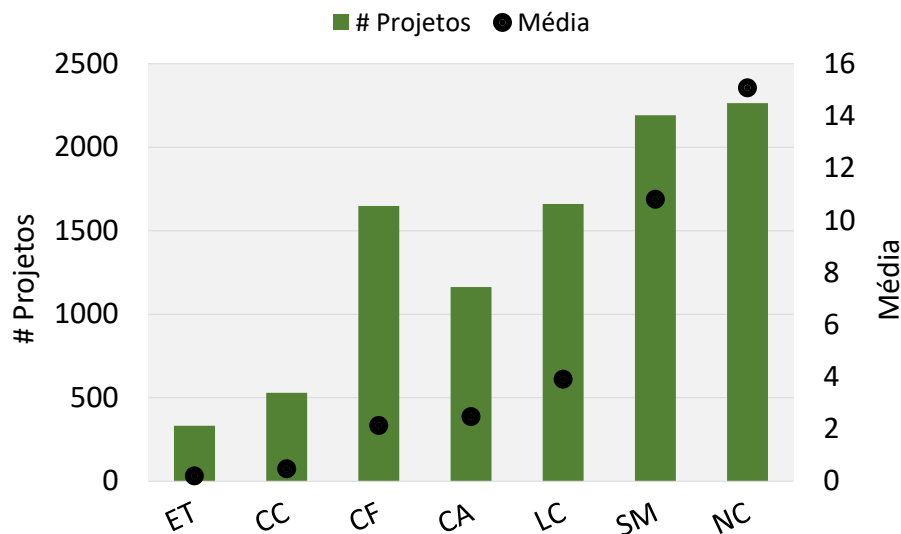


Figura 4.4: Número de projetos preditos contendo cada categoria de informação, e média de parágrafos em determinada categoria por projeto.

Legenda: ET (Escolher a tarefa), CC (Contactar a comunidade), CF (Compreender o fluxo de contribuição), CA (Construir o ambiente de trabalho), LC (Lidar com o código), SM (Submeter mudanças), NC (Nenhuma categoria identificada).

projetos por número de categorias atendidas, equivalente à 324 projetos. Abaixo da média, 286 projetos (13%) apresentaram cinco categorias de informação distintas em seus arquivos de documentação, 165 projetos (7%) apenas uma categoria, e 65 projetos (3%) todas as seis categorias de informação analisadas. Diferente do que havia sido observado na análise qualitativa, onde todos os arquivos continham no mínimo uma categoria de informação, para o conjunto de projetos preditos, 14 arquivos (1%) não apresentaram nenhuma categoria. Em uma rápida análise destes 14 arquivos sem categoria de informação, foi constatado que todos continham informação sobre o processo de contribuição, embora o classificador não tenha identificado nenhuma das categorias propostas em seus parágrafos.

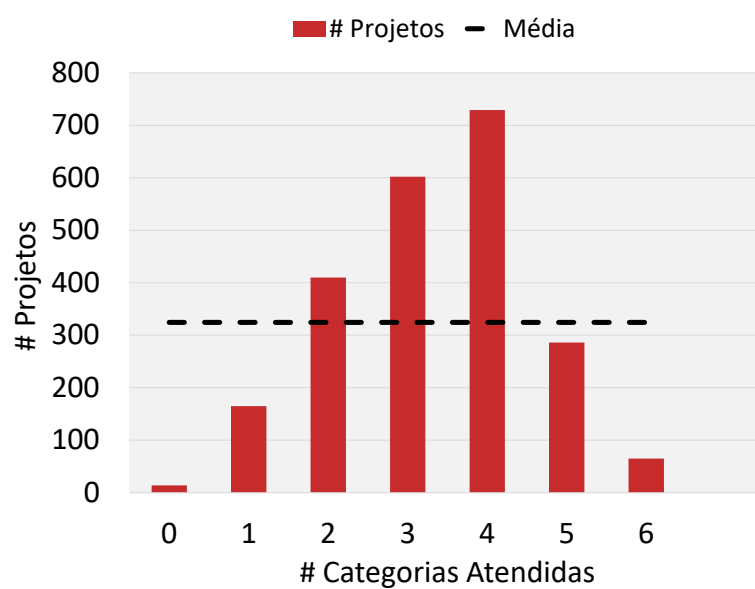


Figura 4.5: Número de categorias de informação atendidas por projeto predito, e linha média de 324 projetos por número de categorias atendidas.

Capítulo 5

Ameaças à validade

5.0.1 Generalidade dos resultados

5.0.2 Representatividade da perspectiva dos contribuidores

5.0.3 Cobertura das categorias de informação

Capítulo 6

Conclusão

6.1 Discussão

6.1.1 Falta de informação essencial para novatos

6.1.2 A maioria dos arquivos são focados nas etapas finais da contribuição

6.2 Considerações finais

Referências

- [ABERDOUR 2007] M. ABERDOUR. “Achieving quality in open-source software”. Em: *IEEE Software* 24.1 (2007), pgs. 58–64. DOI: [10.1109/MS.2007.2](https://doi.org/10.1109/MS.2007.2) (citado na pg. 11).
- [AGHAJANI *et al.* 2019] Emad AGHAJANI *et al.* “Software documentation issues unveiled”. Em: *Proceedings of the 41st International Conference on Software Engineering*. ICSE ’19. Montreal, Quebec, Canada: IEEE Press, 2019, pgs. 1199–1210. DOI: [10.1109/ICSE.2019.00122](https://doi.org/10.1109/ICSE.2019.00122). URL: <https://doi.org/10.1109/ICSE.2019.00122> (citado nas pgs. 1, 10).
- [BORGES *et al.* 2016] Hudson BORGES, Andre HORA e Marco Tulio VALENTE. “Understanding the factors that impact the popularity of github repositories”. Em: *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE. 2016, pgs. 334–344 (citado na pg. 16).
- [BIRD 2006] Steven BIRD. “Nltk: the natural language toolkit”. Em: *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 2006, pgs. 69–72 (citado na pg. 22).
- [BONACCORSO 2017a] Giuseppe BONACCORSO. Packt Publishing, 2017. ISBN: 978-1-78588-962-2. URL: <https://app.knovel.com/hotlink/toc/id:kpMLA00001/machine-learning-algorithms/machine-learning-algorithms> (citado na pg. 24).
- [BONACCORSO 2017b] Giuseppe BONACCORSO. 12.2.2 *Stopword Removal*. Packt Publishing, 2017. ISBN: 978-1-78588-962-2. URL: <https://app.knovel.com/hotlink/khtml/id:kt011DN9A6/machine-learning-algorithms/stopword-removal> (citado na pg. 22).
- [BONACCORSO 2017c] Giuseppe BONACCORSO. 2.1.1.1 *one-vs-all*. 2017. URL: <https://app.knovel.com/hotlink/khtml/id:kt011DN5UB/machine-learning-algorithms/one-vs-all> (citado na pg. 25).
- [BONACCORSO 2017d] Giuseppe BONACCORSO. 3.6 *feature selection and filtering*. 2017. URL: <https://app.knovel.com/hotlink/khtml/id:kt011DN6C1/machine-learning-algorithms/feature-selection-filtering> (citado na pg. 23).

- [BONACCORSI e ROSSI 2003] Andrea BONACCORSI e Cristina ROSSI. “Why open source software can succeed”. Em: *Research policy* 32.7 (2003), pgs. 1243–1258 (citado na pg. 1).
- [BRIAND 2003] L. C. BRIAND. “Software documentation: how much is enough?” Em: *Seventh European Conference on Software Maintenance and Reengineering, 2003. Proceedings.* 2003, pgs. 13–15. DOI: [10.1109/CSMR.2003.1192406](https://doi.org/10.1109/CSMR.2003.1192406) (citado nas pgs. 1, 10).
- [BROWNLEE 2019] Jason BROWNLEE. “How to choose a feature selection method for machine learning”. Em: *Machine Learning Mastery* 10 (2019) (citado na pg. 23).
- [CAMPESATO 2021] Oswald CAMPESATO. *2.7 analyzing classifiers (optional)*. 2021. URL: <https://app.knovel.com/hotlink/khtml/id:kt012NCCW3/python-3-data-analytics/analyzing-classifiers> (citado na pg. 25).
- [CANFORA *et al.* 2012] Gerardo CANFORA, Massimiliano DI PENTA, Rocco OLIVETO e Sebastiano PANICHELLA. “Who is going to mentor newcomers in open source projects?” Em: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM. 2012, pg. 44 (citado na pg. 11).
- [CHANG e LIN 2008] Yin-Wen CHANG e Chih-Jen LIN. “Feature ranking using linear svm”. Em: *Causation and prediction challenge*. PMLR. 2008, pgs. 53–64 (citado na pg. 32).
- [CROWSTON *et al.* 2012] Kevin CROWSTON, Kangning WEI, James HOWISON e Andrea WIGGINS. “Free/libre open-source software development: what we know and what we do not know”. Em: *ACM Computing Surveys (CSUR)* 44.2 (2012), pg. 7 (citado na pg. 1).
- [CUBRANIC *et al.* 2005] D. CUBRANIC, G. C. MURPHY, J. SINGER e K. S. BOOTH. “Hipikat: a project memory for software development”. Em: *IEEE Transactions on Software Engineering* 31.6 (2005), pgs. 446–465. DOI: [10.1109/TSE.2005.71](https://doi.org/10.1109/TSE.2005.71) (citado na pg. 11).
- [DAHLANDER e MAGNUSSON 2008] Linus DAHLANDER e Mats MAGNUSSON. “How do firms make use of open source communities?” Em: *Long Range Planning* 41.6 (2008), pgs. 629–649. ISSN: 0024-6301. DOI: <https://doi.org/10.1016/j.lrp.2008.09.003>. URL: <http://www.sciencedirect.com/science/article/pii/S0024630108000836> (citado na pg. 8).
- [FORWARD e LETHBRIDGE 2002] Andrew FORWARD e Timothy C. LETHBRIDGE. “The relevance of software documentation, tools and technologies: a survey”. Em: *Proceedings of the 2002 ACM Symposium on Document Engineering*. DocEng ’02. McLean, Virginia, USA: ACM, 2002, pgs. 26–33. ISBN: 1-58113-594-7. DOI: [10.1145/585058.585065](https://doi.org/10.1145/585058.585065). URL: <http://doi.acm.org/10.1145/585058.585065> (citado na pg. 10).
- [FRONCHETTI *et al.* 2019] Felipe FRONCHETTI, Igor WIESE, Gustavo PINTO e Igor STEINMACHER. “What attracts newcomers to onboard on oss projects? tl; dr: popularity”.

- Em: *IFIP International Conference on Open Source Systems*. Springer. 2019, pgs. 91–103 (citado nas pgs. 2, 15).
- [FUGGETTA 2003] ALFONSO FUGGETTA. “Open source software—an evaluation”. Em: *Journal of Systems and Software* 66.1 (2003), pgs. 77–90. ISSN: 0164-1212. DOI: [https://doi.org/10.1016/S0164-1212\(02\)00065-1](https://doi.org/10.1016/S0164-1212(02)00065-1). URL: <http://www.sciencedirect.com/science/article/pii/S0164121202000651> (citado na pg. 5).
- [C. GACEK e B. ARIEF 2004] C. GACEK e B. ARIEF. “The many meanings of open source”. Em: *IEEE Software* 21.1 (2004), pgs. 34–40. DOI: [10.1109/MS.2004.1259206](https://doi.org/10.1109/MS.2004.1259206) (citado na pg. 11).
- [Cristina GACEK e Budi ARIEF 2004] CRISTINA GACEK e BUDI ARIEF. “The many meanings of open source”. Em: *IEEE Softw.* 21.1 (jan. de 2004), pgs. 34–40. ISSN: 0740-7459. DOI: [10.1109/MS.2004.1259206](https://doi.org/10.1109/MS.2004.1259206). URL: <http://dx.doi.org/10.1109/MS.2004.1259206> (citado na pg. 5).
- [GITHUB s.d.] GITHUB. *GitHub Flavored Markdown Specs: Paragraphs*. [Accessed on Aug-29-2022]. URL: <https://github.github.com/gfm/#paragraphs> (citado na pg. 18).
- [GITHUB 2020] GITHUB. *GitHub Octoverse*. [Accessed on Jul-1-2020]. 2020. URL: <https://octoverse.github.com/> (citado na pg. 16).
- [GITHUB 2022] GITHUB. *GitHub API*. [Accessed on Out-03-2022]. 2022. URL: <https://docs.github.com/en/rest> (citado na pg. 16).
- [GOOGLE GLOSSARY 2019] GOOGLE GLOSSARY. *Definition of Generalization*. <https://developers.google.com/machine-learning/glossary#generalization>, Last accessed on 20-08-2019. 2019 (citado na pg. 26).
- [HAIDUC *et al.* 2010] S. HAIDUC, J. APONTE, L. MORENO e A. MARCUS. “On the use of automated text summarization techniques for summarizing source code”. Em: *2010 17th Working Conference on Reverse Engineering*. 2010, pgs. 35–44. DOI: [10.1109/WCRE.2010.13](https://doi.org/10.1109/WCRE.2010.13) (citado nas pgs. 1, 13).
- [HANNEBAUER e GRUHN 2017] CHRISTOPH HANNEBAUER e VOLKER GRUHN. “On the relationship between newcomer motivations and contribution barriers in open source projects”. Em: *Proceedings of the 13th International Symposium on Open Collaboration*. OpenSym ’17. Galway, Ireland: ACM, 2017, 2:1–2:10. ISBN: 978-1-4503-5187-4. DOI: [10.1145/3125433.3125446](https://doi.org/10.1145/3125433.3125446). URL: <http://doi.acm.org/10.1145/3125433.3125446> (citado nas pgs. 8, 16).
- [HÖST e ORUČEVIĆ-ALAGIĆ 2011] MARTIN HÖST e ALMA ORUČEVIĆ-ALAGIĆ. “A systematic review of research on open source software in commercial software product development”. Em: *Information and Software Technology* 53.6 (2011), pgs. 616–624 (citado na pg. 6).

- [KAJKO-MATTSSON 2005] Mira KAJKO-MATTSSON. “A survey of documentation practice within corrective maintenance”. Em: *Empirical Software Engineering* 10.1 (2005), pgs. 31–55. ISSN: 1573-7616. DOI: [10.1023/B:LIDA.0000048322.42751.ca](https://doi.org/10.1023/B:LIDA.0000048322.42751.ca). URL: <https://doi.org/10.1023/B:LIDA.0000048322.42751.ca> (citado na pg. 10).
- [KIDO *et al.* 2014] Guilherme Sakaji KIDO, Sylvio Barbon JUNIOR e Stella Naomi MORIGUCHI. *Comparação entre TF-IDF e LSI para pesagem de termos em micro-blog*. 2014 (citado na pg. 22).
- [KLOSTERMAN 2019] Stephen KLOSTERMAN. *6.3.5 exercise 24: building a multiclass classification model for imputation*. 2019. URL: <https://app.knovel.com/hotlink/khtml/id:kt011YZ8C1/data-science-projects/exercise-24-building> (citado na pg. 24).
- [KRAUT *et al.* 2012] Robert E KRAUT, Moira BURKE, John RIEDL e Paul RESNICK. *The challenges of dealing with newcomers*. 2012 (citado na pg. 8).
- [KROGH *et al.* 2003] Georg von KROGH, Sebastian SPAETH e Karim R LAKHANI. “Community, joining, and specialization in open source software innovation: a case study”. Em: *Research Policy* 32.7 (2003). Open Source Software Development, pgs. 1217–1241. ISSN: 0048-7333. DOI: [https://doi.org/10.1016/S0048-7333\(03\)00050-7](https://doi.org/10.1016/S0048-7333(03)00050-7). URL: <http://www.sciencedirect.com/science/article/pii/S0048733303000507> (citado nas pgs. 7, 8, 16).
- [LANTZ 2013] Brett LANTZ. *4.2.3 Data Preparation - Processing Text Data for Analysis*. Packt Publishing, 2013. ISBN: 978-1-78216-214-8. URL: <https://app.knovel.com/hotlink/khtml/id:kt00U5RBR6/machine-learning-with/data-preparation-processing> (citado na pg. 21).
- [LOPER e BIRD 2002] Edward LOPER e Steven BIRD. “Nltk: the natural language toolkit”. Em: *arXiv preprint cs/0205028* (2002) (citado na pg. 22).
- [LI *et al.* 2016] B. LI, C. VENDOME, M. LINARES-VÁSQUEZ, D. POSHYVANYK e N. A. KRAFT. “Automatically documenting unit test cases”. Em: *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2016, pgs. 341–352. DOI: [10.1109/ICST.2016.30](https://doi.org/10.1109/ICST.2016.30) (citado na pg. 13).
- [MALHEIROS *et al.* 2012] Y. MALHEIROS, A. MORAES, C. TRINDADE e S. MEIRA. “A source code recommender system to support newcomers”. Em: *2012 IEEE 36th Annual Computer Software and Applications Conference*. 2012, pgs. 19–24. DOI: [10.1109/COMPSAC.2012.11](https://doi.org/10.1109/COMPSAC.2012.11) (citado na pg. 11).
- [MAO *et al.* 2017] Ke MAO, Licia CAPRA, Mark HARMAN e Yue JIA. “A survey of the use of crowdsourcing in software engineering”. Em: *Journal of Systems and Software* 126 (2017), pgs. 57–84 (citado na pg. 34).
- [Audris MOCKUS *et al.* 2000] Audris MOCKUS, Roy T. FIELDING e James HERBSLEB. “A case study of open source software development: the apache server”. Em: *Proceedings of the 22Nd International Conference on Software Engineering*. ICSE '00.

- Limerick, Ireland: ACM, 2000, pgs. 263–272. ISBN: 1-58113-206-9. DOI: [10.1145/337180.337209](https://doi.org/10.1145/337180.337209). URL: <http://doi.acm.org/10.1145/337180.337209> (citado na pg. 6).
- [NAKAKOJI *et al.* 2002] Kumiyo NAKAKOJI, Yasuhiro YAMAMOTO, Yoshiyuki NISHINAKA, Kouichi KISHIDA e Yunwen YE. “Evolution patterns of open-source software systems and communities”. Em: *Proceedings of the International Workshop on Principles of Software Evolution*. IWPSE '02. Orlando, Florida: ACM, 2002, pgs. 76–85. ISBN: 1-58113-545-9. DOI: [10.1145/512035.512055](https://doi.org/10.1145/512035.512055). URL: <http://doi.acm.org/10.1145/512035.512055> (citado na pg. 7).
- [OWHADI-KARESHK *et al.* 2019] Moein OWHADI-KARESHK, Sarah NADI e Julia RUBIN. “Predicting merge conflicts in collaborative software development”. Em: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, pgs. 1–11 (citado na pg. 25).
- [PANICHELLA *et al.* 2015] S. PANICHELLA *et al.* “How can i improve my app? classifying user reviews for software maintenance and evolution”. Em: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015, pgs. 281–290. DOI: [10.1109/ICSM.2015.7332474](https://doi.org/10.1109/ICSM.2015.7332474) (citado nas pgs. 15, 24, 26).
- [PANICHELLA 2015] S. PANICHELLA. “Supporting newcomers in software development projects”. Em: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015, pgs. 586–589. DOI: [10.1109/ICSM.2015.7332519](https://doi.org/10.1109/ICSM.2015.7332519) (citado na pg. 13).
- [PINTO, DIAS *et al.* 2018] Gustavo PINTO, Luiz Felipe DIAS e Igor STEINMACHER. “Who gets a patch accepted first?: comparing the contributions of employees and volunteers”. Em: *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2018, pgs. 110–113 (citado na pg. 6).
- [PEDREGOSA *et al.* 2011] F. PEDREGOSA *et al.* “Scikit-learn: machine learning in Python”. Em: *Journal of Machine Learning Research* 12 (2011), pgs. 2825–2830 (citado na pg. 24).
- [PINTO, Igor STEINMACHER *et al.* 2018] Gustavo PINTO, Igor STEINMACHER, Luiz Felipe DIAS e Marco GEROSA. “On the challenges of open-sourcing proprietary software projects”. Em: *Empirical Software Engineering* 23.6 (2018), pgs. 3221–3247. ISSN: 1573-7616. DOI: [10.1007/s10664-018-9609-6](https://doi.org/10.1007/s10664-018-9609-6). URL: <https://doi.org/10.1007/s10664-018-9609-6> (citado na pg. 6).
- [PRANA *et al.* 2019] Gede Artha Azriadi PRANA, Christoph TREUDE, Ferdian THUNG, Thushari ATAPATTU e David LO. “Categorizing the content of github readme files”. Em: *Empirical Software Engineering* 24.3 (2019), pgs. 1296–1327. ISSN: 1573-7616. DOI: [10.1007/s10664-018-9660-3](https://doi.org/10.1007/s10664-018-9660-3). URL: <https://doi.org/10.1007/s10664-018-9660-3> (citado nas pgs. 22, 24).
- [QURESHI e FANG 2011] Israr QURESHI e Yulin FANG. “Socialization in open source software projects: a growth mixture modeling approach”. Em: *Organizational Research*

- Methods* 14.1 (2011), pgs. 208–238. DOI: [10.1177/1094428110375002](https://doi.org/10.1177/1094428110375002). eprint: <https://doi.org/10.1177/1094428110375002>. URL: <https://doi.org/10.1177/1094428110375002> (citado na pg. 8).
- [RAYMOND 1999] Eric RAYMOND. “The cathedral and the bazaar”. Em: *Knowledge, Technology & Policy* 12.3 (1999), pgs. 23–49. ISSN: 1874-6314. DOI: [10.1007/s12130-999-1026-0](https://doi.org/10.1007/s12130-999-1026-0). URL: <https://doi.org/10.1007/s12130-999-1026-0> (citado na pg. 6).
- [REID *et al.* 2022] Brittany REID, Markus WAGNER, Marcelo D’AMORIM e Christoph TREUDE. “Software engineering user study recruitment on prolific: an experience report”. Em: *arXiv preprint arXiv:2201.05348* (2022) (citado na pg. 34).
- [ROBILLARD 2009] M. P. ROBILLARD. “What makes apis hard to learn? answers from developers”. Em: *IEEE Software* 26.6 (2009), pgs. 27–34. DOI: [10.1109/MS.2009.193](https://doi.org/10.1109/MS.2009.193) (citado na pg. 10).
- [SANTOS *et al.* 2021] Fabio SANTOS *et al.* “Can i solve it? identifying apis required to complete oss tasks”. Em: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE. 2021, pgs. 346–257 (citado na pg. 25).
- [SOUZA *et al.* 2005] Sergio Cozzetti B. de SOUZA, Nicolas ANQUETIL e Káthia M. de OLIVEIRA. “A study of the documentation essential to software maintenance”. Em: *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*. SIGDOC ’05. Coventry, United Kingdom: ACM, 2005, pgs. 68–75. ISBN: 1-59593-175-9. DOI: [10.1145/1085313.1085331](https://doi.org/10.1145/1085313.1085331). URL: <http://doi.acm.org/10.1145/1085313.1085331> (citado na pg. 10).
- [STALLMAN 2002] Richard STALLMAN. “What is free software”. Em: *Free Society: Selected Essays of* 23 (2002). URL: <https://www.gnu.org/philosophy/free-sw.html> (citado na pg. 5).
- [I. STEINMACHER, CHAVES *et al.* 2014] I. STEINMACHER, A. P. CHAVES, T. U. CONTE e M. A. GEROSA. “Preliminary empirical identification of barriers faced by newcomers to open source software projects”. Em: *2014 Brazilian Symposium on Software Engineering*. 2014, pgs. 51–60. DOI: [10.1109/SBES.2014.9](https://doi.org/10.1109/SBES.2014.9) (citado nas pgs. 8–12, 15).
- [I. STEINMACHER, WIESE *et al.* 2014] I. STEINMACHER, Igor Scaliante WIESE, Tayana CONTE, Marco Aurélio GEROSA e David REDMILES. “The hard life of open source software project newcomers”. Em: *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE 2014. Hyderabad, India: ACM, 2014, pgs. 72–78. ISBN: 978-1-4503-2860-9. DOI: [10.1145/2593702.2593704](https://doi.org/10.1145/2593702.2593704). URL: <http://doi.acm.org/10.1145/2593702.2593704> (citado nas pgs. 1, 8).
- [I. STEINMACHER, CONTE *et al.* 2016] I. STEINMACHER, T. U. CONTE, C. TREUDE e M. A. GEROSA. “Overcoming open source project entry barriers with a portal for newcomers”. Em: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016, pgs. 273–284. DOI: [10.1145/2884781.2884806](https://doi.org/10.1145/2884781.2884806) (citado nas pgs. 15, 16).

- [IGOR STEINMACHER *et al.* 2018] Igor STEINMACHER, Christoph TREUDE e Marco Aurelio GEROSA. “Let me in: guidelines for the successful onboarding of newcomers to open source projects”. Em: *IEEE Software* 36.4 (2018), pgs. 41–49 (citado na pg. 16).
- [TANTITHAMTHAVORN *et al.* 2018] Chakkrit TANTITHAMTHAVORN, Shane MCINTOSH, Ahmed E HASSAN e Kenichi MATSUMOTO. “The impact of automated parameter optimization on defect prediction models”. Em: *IEEE Transactions on Software Engineering* 45.7 (2018), pgs. 683–711 (citado na pg. 33).
- [THANAKI 2017] Jalaj THANAKI. *4.3 Basic Preprocessing*. Packt Publishing, 2017. ISBN: 978-1-78712-142-3. URL: <https://app.knovel.com/hotlink/khtml/id:kt011DM3U2/python-natural-language/basic-preprocessing> (citado na pg. 21).
- [UDDIN e ROBILLARD 2015] G. UDDIN e M. P. ROBILLARD. “How api documentation fails”. Em: *IEEE Software* 32.4 (2015), pgs. 68–75. DOI: [10.1109/MS.2014.80](https://doi.org/10.1109/MS.2014.80) (citado na pg. 10).
- [WANG e SARMA 2011] Jianguo WANG e Anita SARMA. “Which bug should i fix: helping new developers onboard a new project”. Em: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE ’11. Waikiki, Honolulu, HI, USA: ACM, 2011, pgs. 76–79. ISBN: 978-1-4503-0576-1. DOI: [10.1145/1984642.1984661](https://doi.org/10.1145/1984642.1984661). URL: <http://doi.acm.org/10.1145/1984642.1984661> (citado na pg. 11).
- [ZHI *et al.* 2015] Junji ZHI *et al.* “Cost, benefits and quality of software development documentation: a systematic mapping”. Em: *Journal of Systems and Software* 99 (2015), pgs. 175–198. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2014.09.042>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121214002131> (citado na pg. 10).
- [ZHOU e A. MOCKUS 2015] M. ZHOU e A. MOCKUS. “Who will stay in the floss community? modeling participant’s initial behavior”. Em: *IEEE Transactions on Software Engineering* 41.1 (2015), pgs. 82–99. ISSN: 0098-5589. DOI: [10.1109/TSE.2014.2349496](https://doi.org/10.1109/TSE.2014.2349496). URL: <https://ieeexplore.ieee.org/document/6880395> (citado na pg. 8).
- [ZHONG e SU 2013] Hao ZHONG e Zhendong SU. “Detecting api documentation errors”. Em: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA ’13. Indianapolis, Indiana, USA: ACM, 2013, pgs. 803–816. ISBN: 978-1-4503-2374-1. DOI: [10.1145/2509136.2509523](https://doi.org/10.1145/2509136.2509523). URL: <http://doi.acm.org/10.1145/2509136.2509523> (citado nas pgs. 1, 13).