Classificação de repositórios *open source* com foco na experiência de iniciantes*

Gustavo Gewehr Soares[1], **Tiago Rios da Rocha**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - *Campus* Ibirubá
Rua Nelsi Ribas Fritsch, 1111 – CEP: 98200-000 – Ibirubá – RS – Brasil

Abstract. *The study aims to categorize open source repositories that facilitate the onboarding of novice developers, using data from GHtorrent and machine learning algorithms such as Artificial Neural Networks (ANN), K-Nearest Neighbors (KNN), and Support Vector Machines (SVM). The research identified that recent projects with high commit activity and larger communities are more attractive to new contributors. Neural Networks showed the best performance with 77% accuracy but demonstrated limitations in identifying less welcoming projects. The work highlights the importance of facilitating novice integration to strengthen open source communities and proposes future improvements, such as personalized recommendations.*

Resumo. *O estudo visa categorizar repositórios open source que facilitam a integração de desenvolvedores iniciantes, utilizando dados do GHtorrent e algoritmos de aprendizado de máquina, como Artificial Neural Network (ANN), K-Nearest Neighbors (KNN) e Support Vector Machines (SVM). A pesquisa identificou que projetos recentes, com alta atividade de commits e comunidades maiores, são mais atrativos para novos colaboradores. Redes Neurais apresentaram melhor desempenho, com 77% de acurácia, mas demonstraram limitações na identificação de projetos menos receptivos. O trabalho destaca a importância de facilitar a inclusão de iniciantes para fortalecer comunidades open source e propõe aprimoramentos futuros como a personalização das recomendações.*

1. Introdução

Os *softwares* livres (*open source*) diferenciam-se dos demais *softwares* pelo seu modelo único de desenvolvimento, no qual o código fonte é aberto para qualquer pessoa e as contribuições podem ser efetuadas por qualquer interessado em colaborar. Eles são impulsionados por uma grande comunidade de usuários, desenvolvedores e gestores. Essas características acarretaram os projetos de código aberto se tornarem a infraestrutura da sociedade moderna. Uma das principais aplicações para controle de versionamento e contribuições de *softwares open source* é o *GitHub* (XIAO et al., 2022).

O *GitHub* é um serviço online que oferece controle de versões e gerenciamento de código. Ele permite aos seus usuários a criação de repositórios, os quais comportam e disponibilizam o código fonte do *software*. Em janeiro de 2023 o GitHub (2023a) reportou ter atingido a marca de mais de cem milhões de desenvolvedores usuários da ferramenta. No mesmo ano, em novembro, a plataforma divulgou o montante de quatrocentos e vinte

*Trabalho de Conclusão de Curso (TCC) do curso Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul – Câmpus Ibirubá. 2025.

milhões de projetos que utilizam os seus serviços. Desses, duzentos e oitenta e quatro milhões são repositórios públicos, ou seja, que possuem código aberto. Ainda relatou um crescimento de quantidade de repositórios públicos de vinte e dois porcento ao ano. Gerando um total de quatro bilhões e meio de contribuições, dispersos por todos os projetos, no mesmo ano (GITHUB, 2023b).

Os projetos de código aberto dependem diretamente do engajamento constante e longevo dos seus desenvolvedores. Portanto, para manter o ambiente *open source* saudável, é essencial sustentar o número de contribuintes através da integração de novos desenvolvedores. Mesmo tendo um alcance global, os projetos de código aberto ainda possuem dificuldade em se manter, uma vez que os seus contribuintes são voluntários (FORD; SHRESTHA; ZIMMERMANN, 2022). Portanto, as contribuições são distribuídas em um grande espectro de frequência. Dessa forma, é crucial para esses *softwares* integrarem uma grande quantidade de desenvolvedores capazes de contribuírem por um longo período.

As comunidades de livre colaboração dependem de novas contribuições para prosperarem. Inclusive há a perspectiva de que novos contribuintes sejam fonte de inovação tecnológica. Porém, uma das maiores dificuldades dos *softwares open source* é atrair e manter novas contribuições. Mesmo sendo a chave para a longevidade da comunidade, a integração bem sucedida de desenvolvedores iniciantes à projetos *open source* pode ser bem problemática (XIAO et al., 2022). Por isso estudos referentes ao ingresso de iniciantes se fazem importantes.

Steinmacher et al. (2015b) enfatizam que frequentemente os iniciantes encontram barreiras ao tentarem contribuir em projetos *open source* e muitas vezes elas os levam a desistirem de contribuir com projetos de código aberto. A falha de uma primeira boa impressão pode ocasionar um longo período de tempo até que o desenvolvedor dê outra chance a projetos de código aberto. Por causa disso, Steinmacher et al. (2015b) defende que um dos maiores desafios dos projetos de software livre é reduzir essas barreiras de entrada. Uma maneira de compreender melhor o comportamento de iniciantes diante das barreiras

O objetivo do presente trabalho é utilizar um modelo analítico, que utiliza de técnicas de aprendizado de máquina, para a classificação de repositórios de acordo com a experiência de iniciantes durante o seu processo de integração, acarretando facilitar a contribuição de uma pessoa iniciante no desenvolvimento de softwares *open source*. Já que classificar repositórios altamente qualificados para primeiras contribuições diminui as barreiras de entrada na comunidade *open source*. Há a possibilidade de que os repositórios recebam maiores quantidades de contribuições e contribuições de mais longo prazo, assim fortalecendo a comunidade, aprimorando a capacidade dos projetos de se manterem ativos e prosperarem. Com isso em mente,

O reconhecimento de padrões utiliza algoritmos para identificar regularidades nos dados e aplicá-las em tarefas como classificação (BISHOP, 2016). Nesse sentido, o aprendizado de máquina, conforme Géron (2017), é eficaz para problemas complexos e dinâmicos, oferecendo soluções mais simples e adaptáveis em comparação a métodos tradicionais. Dessa forma, este método é adequado para lidar com o problema de categorização de repositórios de acordo com a experiência de iniciantes.

Neste trabalho, construiu-se um modelo analítico baseado em aprendizado de máquina para recomendar repositórios *open source* que favoreçam a integração de desenvolvedores iniciantes. Utilizando dados históricos do GHtorrent e técnicas de classificação supervisionada, como K-Nearest Neighbors (KNN), Support Vector Machines (SVM) e Artificial Neural Networks (ANN), o modelo identificou variáveis-chave que influenciam positivamente a experiência dos novos contribuidores, como o tempo de criação do projeto, a atividade recente de *commits* e o número de membros. Os resultados demonstraram que redes neurais apresentaram o melhor desempenho, com uma acurácia de 77% e 76% de precisão na recomendação de repositórios amigáveis a iniciantes. No entanto, o modelo mostrou limitações na identificação de projetos não amigáveis, indicando oportunidades para aprimoramentos futuros. Esses resultados destacam a relevância de iniciativas que promovam a inclusão de novos colaboradores, contribuindo para o fortalecimento das comunidades *open source* e a sustentabilidade de seus projetos.

Para melhor discutir a classificação de repositórios *open source* de acordo com a experiência de inciantes e apoiar o desenvolvimento de *softwares* de código aberto, este artigo está organizado como segue: 1. Introdução; 2. Referencial Teórico; 3. Trabalhos Correlatos; 4. Metodologia; 5. Compreensão, Aquisição e Preparação dos Dados; 6. Modelagem dos Algoritmos; 7. Análise dos resultados; 8. Conclusão.

2. Referencial Teórico

Para garantir a boa compreensão deste trabalho, a presente seção elucida os conceitos de *software open source*, plataforma *GitHub*, aprendizado de máquina e barreiras de contribuição.

2.1. *Open Source e GitHub*

Segundo Zhang et al. (2017), os *open source*, diferentemente dos outros tipos de softwares, são impulsionados por uma grande comunidade de usuários, desenvolvedores e gestores. Essas pessoas se envolvem nos projetos de *software* livre por interesse próprio, muitos ainda possuem trabalho de tempo integral e somente podem contribuir no seu tempo livre. Elas podem contribuir de forma esporádica ou frequente, mesmo assim existem diversos *open source* de grande sucesso. Dessa forma, é crucial para estes *softwares* encontrarem os desenvolvedores capazes de contribuir. Em contrapartida, projetos adequados podem chamar mais atenção e inspirar os desenvolvedores a contribuírem continuamente. Um serviço muito popular para hospedagem do código fonte e seu compartilhamento é o GitHub.

O *GitHub* é um serviço online que oferece controle de versões e gerenciamento de código. Ele permite a criação de repositórios, nos quais são armazenadas partes do código-fonte de um determinado *software*. Dessa forma, a plataforma acompanha a evolução das diferentes versões do *software* (GITHUB, 2023c). Conforme evidenciado na figura 1 ela é uma plataforma muito popular, contando com mais de 4 bilhões de contribuições em mais de 284 milhões de repositórios *open source* no ano de 2023. Sua popularidade facilita a colaboração e a participação no desenvolvimento de *software*. No *GitHub*, os desenvolvedores podem iniciar projetos próprios, ou seja, criar repositórios, além de contribuir com repositórios de terceiros.

O *GitHub* também possibilita a colaboração no desenvolvimento de *software* distribuído por meio do modelo de *fork* e *pull*. Desenvolvedores podem criar uma cópia

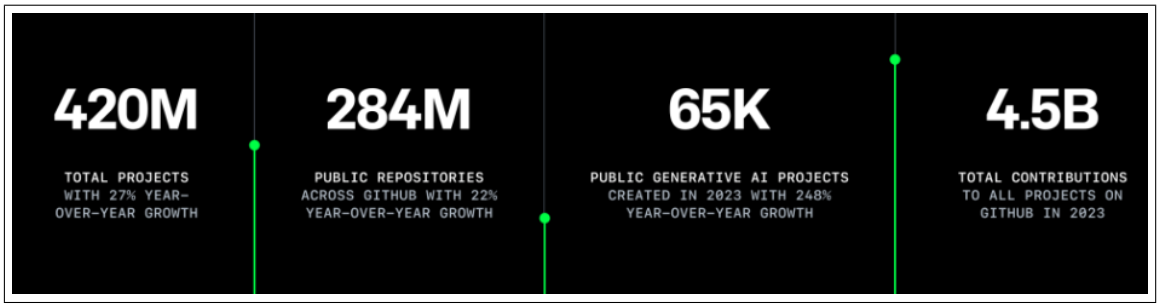

Figura 1. Dados de utilização do GitHub. Fonte: GitHub (2023c)

(*fork*) de qualquer repositório público no *GitHub* e realizar suas próprias alterações. Essas modificações podem então ser enviadas de volta ao repositório original por meio de um *pull request*. O mantenedor do repositório analisará as mudanças e decidirá se as incorpora ao repositório principal (JIANG et al., 2017).

A estrutura social e técnica do *GitHub* permite investigar como contribuições externas influenciam o crescimento dos projetos e como desenvolvedores são promovidos em equipes de código aberto, fornecendo uma base valiosa para compreender a formação de comunidades, as estratégias de promoção e a colaboração distribuída. Porém, as informações do GitHub ainda são de difícil acesso devido a algumas limitações impostas às APIs de usuários.

Uma maneira encontrada por pesquisadores para mitigar as limitações impostas pelo *GitHub* na sua API foi a criação do *GHtorrent*. Ele é um agregador de chaves de acesso da API do *GitHub* que permite que a aplicação fique se servindo dos eventos realizados na plataforma. Consequentemente, ele possui um banco de dados paralelo ao *GitHub* e são disponibilizados para a comunidade realizar download e utilizá-los . Por meio do *GHTorrent*, esses dados são coletados e organizados, possibilitando análises em larga escala de ecossistemas de *software*, redes de colaboração e do impacto das contribuições em projetos abertos (GOUSIOS, 2013a).

2.2. Aprendizado de Máquina

O aprendizado de máquina é um campo de estudo que emprega algoritmos computacionais para identificar automaticamente regularidades nos dados e utilizá-las na tomada de decisões, como a classificação dos dados em diferentes categorias (BISHOP, 2016). Segundo Géron (2017), o aprendizado de máquina tem como objetivo extrair *insights* a partir de problemas complexos e envolvendo grandes quantidades de dados, mostrando-se mais eficaz para questões muito difíceis de resolver por métodos tradicionais ou para aquelas em que não existe um algoritmo conhecido. Em situações que demandariam um extenso ajuste manual ou longas listas de regras, um único algoritmo de aprendizado de máquina pode simplificar o código e alcançar desempenho superior. Dessa forma, problemas complexos para os quais não há uma solução adequada por abordagens tradicionais podem ser resolvidos pelas melhores técnicas de aprendizado de máquina. Além disso, em ambientes dinâmicos, que mudam constantemente, sistemas de aprendizado de máquina podem se adaptar aos novos dados. De maneira a tornar possível o trabalho de aprendizado de máquina é necessário realizar o pré-processamento dos dados.

O pré-processamento de dados é uma etapa fundamental no processo de mine-

ração, pois assegura que os dados brutos estejam organizados e formatados de forma adequada para a análise pelos algoritmos de aprendizagem de máquina. De acordo com Aggarwal (2015), o pré-processamento divide-se em três principais etapas: extração de recursos, na qual os dados são transformados em características significativas, demandando uma análise criteriosa para a seleção dos *features* mais relevantes; limpeza de dados, que corrige erros, além de remover entradas ausentes ou inconsistentes, aprimorando a qualidade do *dataset*; e seleção e transformação de recursos, que elimina elementos irrelevantes e ajusta o espaço de dados, otimizando a análise. Essas fases são indispensáveis para garantir eficácia e precisão nos modelos de aprendizado de máquina.

Os algoritmos de aprendizado de máquina podem ser classificados de acordo com o grau de supervisão necessário. Podendo ser supervisionado, em que a classe objetivo já possui valores conhecidos no treinamento. E não supervisionado em que o algoritmo é treinado sem valores definidos para a classe objetivo. Neste trabalho, utilizou-se apenas o aprendizado supervisionado, no qual utiliza-se um grande *dataset*, chamado *dataset* de treinamento, para ajustar os parâmetros de um modelo adaptável. As categorias presentes nesse *dataset* de treinamento são conhecidas previamente, geralmente por meio de inspeção e rotulagem manuais. (GÉRON, 2017).

Uma vez treinado, o modelo de Aprendizado de Máquina pode categorizar novas instâncias do *dataset* de teste. A capacidade de generalizar – isto é, classificar corretamente exemplos diferentes dos utilizados no treinamento – é um objetivo central no reconhecimento de padrões (BISHOP, 2016). Essa generalização é essencial, pois os dados de treinamento representam apenas uma pequena fração do universo possível de vetores de entrada. O principal objetivo ao aplicar um modelo de aprendizado de máquina é alcançar o melhor desempenho com os dados pré-processados, sendo o resultado esperado um critério determinante na escolha do modelo. Para auxiliar nessa seleção, Vora e Bhatia (2023) organizam os modelos conforme o tipo de problema para o qual cada abordagem é mais eficiente e colocam a classificação como os tipos de modelos mais populares.

A classificação é uma das tarefas mais frequentes no aprendizado de máquina, sendo um exemplo de aprendizagem supervisionada. Ela consiste em examinar as características de um novo objeto e atribuir-lhe uma classe pré-determinada (LINOFF; BERRY, 2011). Um modelo de classificação é aplicado a um *dataset* não classificados com o objetivo de organizá-los, exigindo, para tal, que a *target-feature* seja uma classe (VORA; BHATIA, 2023).

Linoff e Berry (2011) destacam que as tarefas de categorização são as mais comuns na área de mineração de dados, afirmando também que a necessidade de classificação é inerente ao ser humano, pois, para entender e se comunicar sobre o mundo, o homem está constantemente classificando. Neste trabalho serão utilizadas as técnicas Support Vector Machines (SVM), K-nearest-neighbor (KNN) e *Artificial Neural Networks* (ANN), suas respectivas ilustrações estão disponíveis na figura 2.

O modelo de *Support Vector Machines* (SVM) representa os dados de treinamento como pontos em um espaço. Em seguida, traça hiperplanos nesse espaço de modo a separar os pontos, buscando aquele que maximize a distância entre o hiperplano e os pontos mais próximos a ele. Ao término, os pontos de um lado do hiperplano são considerados positivos, enquanto os do lado oposto são negativos (VORA; BHATIA, 2023).

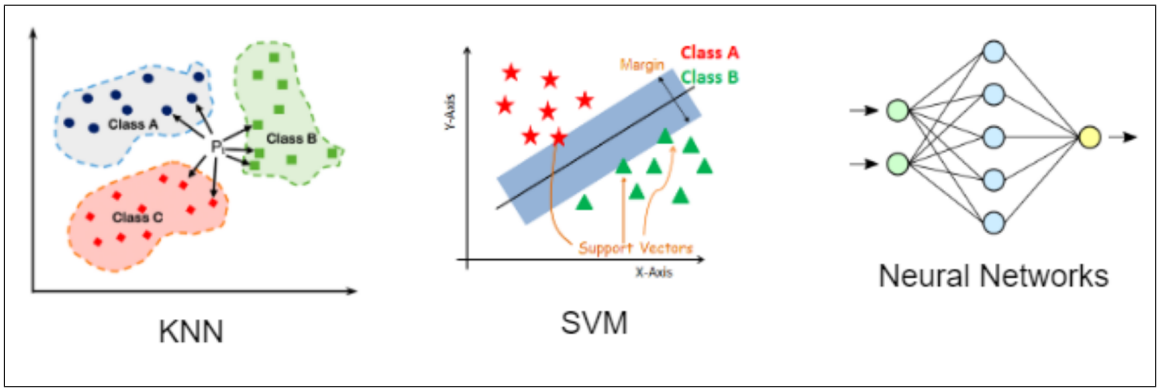

Figura 2. Ilustrações dos algoritmos utilizados. Fonte: Autor

O classificador *K-Nearest Neighbor* (K-NN) também insere os dados de treinamento em um espaço de pontos. Ao receber um novo objeto, o classificador o posiciona nesse espaço e busca os K elementos mais próximos dele no *dataset* de treinamento. Em seguida, verifica quais classes esses vizinhos pertencem, atribuindo ao objeto desconhecido a classe majoritária encontrada (FARIA, 2016).

A Rede Neural Artificial (*Artificial Neural Network ou ANN*) é uma classe de técnicas de Aprendizado de Máquina utilizadas para modelar padrões complexos em conjuntos de dados, empregando múltiplas camadas ocultas e funções de ativação não lineares (VORA; BHATIA, 2023). Ela é composta de unidades básicas que imitam o comportamento dos neurônios biológicos, de forma que cada unidade recebe diversas entradas, processa-as e as combina em um único valor de saída (FARIA, 2016).

2.3. Barreiras de Contribuição

Comunidades de livre colaboração dependem de novas contribuições. Inclusive há a perspectiva de que novos contribuintes sejam fonte de inovação. No entanto, de acordo Steinmacher et al. (2015a) durante o processo de integração, novos colaboradores enfrentam diversas barreiras de entrada, como quebra de expectativa, problemas de configuração e curva de aprendizagem. Wang e Sarma (2011) apresentam que um dos maiores desafios no desenvolvimento de *software* é fazer com que os desenvolvedores se familiarizem rapidamente com o projeto, ou seja, obter o entendimento dos recursos relevantes em um novo projeto é essencial e demanda muito tempo e esforço.

Steinmacher et al. (2015b) infere que os recém-chegados à comunidade *open source* que aspiram fazer alguma contribuição a um projeto, devem possuir uma combinação de conhecimento do domínio, habilidades técnicas e interação social. Essa combinação pode aumentar a probabilidade de uma integração bem-sucedida. Os resultados dessas interações permitirão tanto aos iniciantes quanto aos experientes avaliarem a extensão e a possível ausência de conhecimento que poderia dificultar sua participação eficaz no projeto.

A tarefa de mitigação de barreiras de contribuição não é algo simples de resolver. A quantidade de variáveis que influenciam a experiência de um desenvolvedor entregando sua primeira contribuição e o comportamento do relacionamento entre elas torna este um problema complexo, ou seja, não é trivial resolver, de forma satisfatória, com um algoritmo simples. Para Steinmacher et al. (2015b), as origens das dificuldades de integração

de iniciantes à comunidade de *software* de código aberto podem ser separadas em três categorias. Primeiramente, ele apresenta as barreiras técnicas, que dizem respeito ao conhecimento do indivíduo que pretende contribuir e à dificuldade de conseguir orientação. Depois, ele traz as barreiras sociais, que focam na interação dos novos contribuintes com membros antigos e mantenedores do projeto. E por último, ele expõe as barreiras relacionadas ao produto, que competem a características do projeto, como documentação, comentários de códigos, entre outros. Por fim, ele afirma que mitigar estas barreiras contribui positivamente para a sobrevivência e prosperidade da comunidade *open source*. Os próximos parágrafos discorrem sobre algumas barreiras de contribuição e sua relação com as *features* dos repositórios. São elas: dificuldade em encontrar uma forma de começar; barreira de interação social; barreiras de documentação; e barreiras técnicas:

2.4. Dificuldade em encontrar uma forma de começar:

Em ambientes industriais, para apoiar seus primeiros passos, é comum oferecer um caminho de integração aos novos membros. No entanto, em projetos de código aberto, que dependem de voluntários, essa não é uma prática comum. As comunidades frequentemente esperam que os novos participantes encontrem suas próprias tarefas, o que pode ser intimidador para aqueles que ainda não compreendem a dinâmica do projeto. Encontrar a tarefa apropriada para trabalhar foi classificado como um problema, pois novos desenvolvedores têm dificuldade em encontrar falhas que sejam do seu interesse, que correspondam às suas habilidades, que não sejam duplicadas e que sejam importantes para a sua futura comunidade. Para superar essa barreira, é importante que os projetos forneçam suporte mais estruturado, como listas de tarefas para iniciantes e sistemas de mentoria acessíveis (STEINMACHER et al., 2015b).

2.5. Barreira de interação social:

A falta de interação social é uma das dificuldades mais significativas enfrentadas por novos contribuintes em projetos de software de código aberto. Ela se manifesta principalmente pela falta de integração com membros da comunidade, ausência de respostas ou respostas inadequadas às dúvidas e contribuições iniciais dos iniciantes. Essa situação pode levar ao sentimento de desmotivação e exclusão, afetando negativamente a decisão de continuar contribuindo. Estudos mostram que interações sociais positivas, como respostas rápidas e acolhedoras, são essenciais para o engajamento e a retenção de novos participantes. Além disso, construir uma rede de contatos e estabelecer uma identidade no projeto são aspectos fundamentais para que os iniciantes se sintam parte da comunidade e obtenham sucesso em suas contribuições (STEINMACHER et al., 2015b).

2.6. Barreiras de documentação:

A documentação é essencial para a contribuição de iniciantes,pois eles dependem dela para compreender os aspectos técnicos e sociais do projeto. Em muitos projetos de código aberto, as informações existentes estão dispersas em listas de discussão, repositórios de código-fonte, rastreadores de problemas e páginas do projeto. No entanto, tentativas excessivas para resolver isso podem levar ao problema oposto, ocasionando uma documentação excessiva e fontes de informação avassaladoras, acarretando sobrecarga de informações. Do mesmo modo, fornecer documentação desatualizada pode se tornar uma barreira para os novos membros em vez de ajudá-los. Para mitigar essas barreiras, os

projetos devem priorizar a criação e manutenção de documentações claras, atualizadas e organizadas, além de fornecer ferramentas que facilitem a navegação e recomendação de informações relevantes (STEINMACHER et al., 2015b).

2.7. Barreiras técnicas:

As barreiras técnicas têm uma influência negativa significativa no número de contribuições de novos desenvolvedores, pois elas afetam diretamente a capacidade dos iniciantes em interagir com o código para criar uma contribuição. Entre os principais desafios estão a configuração do ambiente local, que pode envolver problemas como a falta de experiência em usar sistemas de controle de versão ou dificuldades em resolver dependências e compilar o projeto. Além disso, a complexidade do código e da arquitetura frequentemente aumentam o tempo para um desenvolvedor iniciante entender o fluxo de lógica, os padrões no código-fonte e a estrutura do diretório do código-fonte, resultando em progresso mais lento do projeto. Para superar essas barreiras, é essencial que os projetos ofereçam suporte técnico robusto, como guias de configuração detalhados, ferramentas automatizadas para o ambiente de desenvolvimento e recursos que facilitem a compreensão da estrutura do código, como visualizações e mapas de dependências (STEINMACHER et al., 2015b).

3. Trabalhos Correlatos

No que se refere ao escopo desse trabalho, quatro estudos se mostraram próximos suficientes ao objetivo deste trabalho para entender melhor o problema. As diversas formas de soluções adotadas e como o presente estudo diferencia-se dos demais estão dispostas no decorrer desta seção.

O estudo de Xiao et al. (2022) tenta utilizar as informações disponíveis no *GHtorrent* e na API do Github a respeito de boas primeiras tarefas (BPF), para treinar o classificador eXtreme Gradient Boosting (XGBoost) a identificar BPFs e o nomeou de RecGFI . O processo de coleta dos dados demorou um mês e utilizou quinze chaves diferentes de acesso à API. Para validar o resultado do algoritmo foi utilizado o método AUC (*Area under the ROC Curve*). Além disso o comparou com outros cinco algoritmos padrões. Como também, realizou uma comparação com comportamentos de contribuições assistidas pelos pesquisadores, em que captou novos dados e aplicou o modelo. Por fim, concluiu que o RecGFI alcançou alta performance, performou melhor que os demais métodos alternativos e revelou observações interessantes sobre as caraterísticas de BPFs. Porém, deixou aberta a possibilidade de implantar o RecGFI para utilização do público.

No trabalho de Liu et al. (2018), os pesquisadores buscavam capturar o padrão complexo de integração em projetos *open source* disponíveis na fonte de dados *GHtorrent*. Para isso, os pesquisadores propuseram a utilização do modelo NNLRank (*Neural Network for List-wise Ranking*) para recomendar projetos que os desenvolvedores são prováveis de produzir contribuição. Para garantir a efetividade deste modelo, a pesquisa compara a performance de mais três modelos alternativos com o modelo gerado no trabalho. Os pesquisadores validaram o algoritmo utilizando MRR (Mean Reciprocal Rank) e MAP (Mean Average Precision). O estudo observou 2044 processos de integração bem sucedidos, que tiveram mais de 6 *commits* após ele, e confirmou a eficiência e eficácia do modelo treinado.

A pesquisa de Sarma et al. (2016) busca criar uma plataforma, chamada *BugExchange*, para auxiliar iniciantes a se integrarem a novos projetos e treiná-los para participarem do ambiente de desenvolvimento de software *open source*. O trabalho se dividiu em duas seções: classificação de tarefas, que utiliza um mecanismo de votação para realizar a catalogação das tarefas; portal FLOSSCoach, para sustentar os primeiros passos dos iniciantes, que ataca as principais barrerias identificadas em trabalhos anteriores. Os pesquisadores avaliaram o resultado da abordagem com estudantes de engenharia de software de múltiplas universidades utilizando diários de estudos para monitorar o progresso dos estudantes, questionários de eficácia própria e pré e pós estudos, juntamente com entrevistas com os instrutores e membros dos projetos. Concluiram que a plataforma *BugExchange*, possivelmente reduz a quantidade de desistências e incentiva mais contribuições casuais para os projetos.

Os pesquisadores do estudo de Ford, Shrestha e Zimmermann (2022) construíram um sistema para recomendar contribuições em projetos com impacto social. Para isso, eles utilizaram como fonte de dados o *GithubAPI* e o *GHtorrent*. Para observar quatro sinais principais, baseado na alta visibilidade dos *features* que os desenvolvedores já utilizam para basear suas decisões, a experiência em linguagens de programação anterior do usuário, os tópicos dos projetos já contribuídos por ele, a relação dele com outros colaboradores e seu histórico de contribuições recentes. O sistema construído no estudo passou por uma validação *offline*, comparando o resultado com dados de teste, juntamente com três entrevistas.

Pode-se comparar os demais trabalhos apresentados anteriormente, nesta seção com o que o presente projeto realizou, como no Quadro 1. Afim de facilitar o entendimento das diferenças entre eles foram destacadas algumas variáveis. Inciando pela fonte de dados, que busca entender qual seria o método adotado pelos trabalhos para obter os dados que eles se basearam para realizar suas respectivas análises. A variável modelo destaca qual o algoritmo que foi treinado pelo trabalho afim de realizar alguma recomendação para o usuário. O objeto de estudo dá enfoque à principal entidade que engloba as *features* analisadas pela pesquisa para gerar recomendações. A plataforma diz respeito a forma como o resultado do trabalho foi entregue à comunidade para utilização das recomendações. Por último, um identificador se o trabalho estudou o comportamento de desenvolvedores iniciantes ou não.

Trabalho	Fonte dos dados	Modelo	Obj. Estudo	Plataforma	Iniciantes
Xiao et al. (2022)	*GHtorrent*, GitHubAPI	XGBoost	Tarefas	Não há	Sim
Liu et al. (2018)	*GHtorrent*	NNLRank	Repositórios	Não há	Não
Sarma et al. (2016)	Não há	Não há	Tarefas	BugExchange ou FlossCoach	Sim
Ford, Shrestha e Zimmermann (2022)	GithubAPI, *GHtorrent*	Não há	Usuário e Repositórios	Não há	Não
Presente trabalho	*GHtorrent*	Não há	Usuário e Repositórios e Commits	Não há	Sim

Quadro 1. Comparativo dos trabalhos correlatos.

Embora diversas abordagens já almejem criar um sistema de recomendações, nenhuma das elencadas nesta seção ofereceu uma solução que combinasse um algoritmo de inteligência artificial com a perspectiva das características dos repositórios e elencando quais destes seriam os mais receptivos a iniciantes.

4. Metodologia

Esta seção detalha o processo utilizado durante a produção do estudo. A qual foi baseada no método CRISP-DM. Segundo Wirth e Hipp (2000) este método provou ser eficaz, fornecendo um modelo de processo genérico, útil para planejamento, documentação e comunicação. A metodologia está dividida em três partes, conforme pode ser visto na Figura 3, são elas: Compreensão, aquisição e preparação dos dados; Modelagem do algoritmo; e Avaliação do resultado do treinamento.

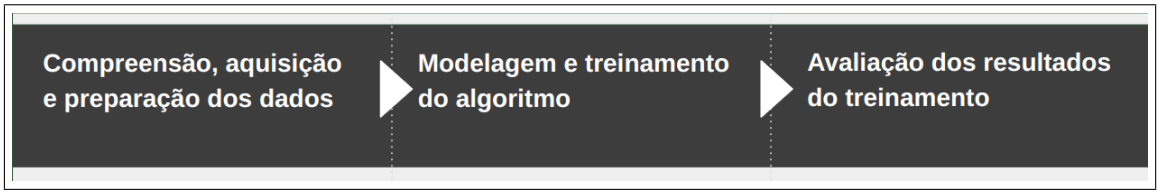

Figura 3. Etapas da metodologia. Fonte: Autor

4.1. Compreensão, aquisição e preparação dos dados

Inicialmente, os esforços se concentram em entender os objetivos do projeto e os requisitos de uma perspectiva de negócios, e então converter esse conhecimento em uma definição de problema de mineração de dados. Ele é seguido pela aquisição de dados que foca na coleta de informações relevantes, exploração de suas propriedades e identificação de problemas de qualidade, como inconsistências e valores ausentes. A preparação dos

dados, por sua vez, transforma esses dados brutos no *dataset* final para modelagem, por meio de limpeza, seleção, criação de *features* e ajustes, de forma iterativa.

A coleta de dados não é trivial e está sujeita a diversas restrições. Posto isto, Cosentino, Izquierdo e Cabot (2016) apontaram em seu estudo que *datasets* curados que replicam os dados do *Github* são as principais fontes de dados utilizadas em estudos envolvendo os dados do *Github*. Dentre todos os métodos de extração de dados estudados por eles, o *GHtorrent*, a APi do *Github* e o *Github Archive* foram os três conjuntos de dados mais utilizados em pesquisas. Portanto, com a finalidade de manter a replicabilidade deste estudo, fora utilizado o *dataset* do *GHtorrent* (GOUSIOS, 2013b).

Após a aquisição dos dados, foi necessário construir o dataset de treinamento dos algoritmos. Isso só é possível pois pode-se inferir mais *features* para o repositório ao condensar as informações distribuídas no dataset original. Para isso, algumas características referentes às barreiras de contribuições foram isoladas. Este passo está mais detalhado na seção 5. Neste momento, é propício realizar a separação de uma porção do *dataset* resultante com a finalidade de, posteriormente, validar os resultados.

Para a fase de preparação de dados foi utilizada a linguagem de programação *Python* pois Raschka (2015) traz que esta é a linguagem de programação mais popular para a ciência de dados. *Numpy* e *Pandas* são bibliotecas utilizadas para performar diversas tarefas de Aprendizado de Máquina. *Numpy* disponibiliza um objeto *array* poderoso, e muita utilidade para operações de álgebra linear. *Pandas* é a biblioteca mais útil para preprocessamento e preparação de dados para Aprendizado de Máquina, principalmente pelo fornecimento do tipo de dados chamado *Dataframe* que facilita o processamento dos dados (VORA; BHATIA, 2023).

4.2. Modelagem do algoritmo

A modelagem é a fase em que técnicas de mineração de dados são aplicadas para construir modelos que atendam aos objetivos definidos, ajustando parâmetros para otimizar o desempenho. Algumas técnicas exigem formatos de dados específicos, criando uma relação próxima com a preparação dos dados, já que problemas ou ideias para novos *features* frequentemente surgem durante a modelagem. Portanto, após preparar a base de dados é possível realizar a modelagem do algoritmo. Para isso, utilizando a linguagem de programação *Python* em conjunto com a biblioteca *Scikit-Learn*, a base de dados de treinamento foi exposta aos algoritmos apontados como classificadores por Vora e Bhatia (2023), k-Vizinhos Mais Próximos (k-NN), Máquina de Vetores de Suporte (SVM) e Redes Neurais. Estes algoritmos foram selecionados pois apresentaram os melhores resultados no estudo de Liu et al. (2018). Testes rápidos e frequentes foram realizados para que fosse possível ajustar os parâmetros destes algoritmos. Quando os ajustes passaram a fornecer retorno diminuto o algoritmo que teve o melhor resultado foi selecionado para a classificação de repositórios.

O *Scikit-learn* é uma biblioteca *open source* de Aprendizado de Máquina mais populares e acessíveis (RASCHKA, 2015). Ela incorpora uma variedade extensa de algoritmos de Aprendizado de Máquina avançados, destinados a abordar desafios tanto em problemas supervisionados quanto não supervisionados de escala intermediária (PEDREGOSA et al., 2011). .

4.3. Avaliação do resultado do treinamento

Por fim, é necessário avaliar o quão eficaz foi o treinamento do algoritmo de aprendizado de máquina. Foram separados diversos conjuntos de repositórios aleatórios, os quais houveram alguns que receberam uma contribuição inicial. Após isso, o algoritmo deve elencar, dentre os repositórios, quais ele assume como os mais receptivos às primeiras contribuições.

As métricas utilizadas para medir o desempenho dos classificadores foram revocação e precisão. A revocação foi calculada através da quantidade de repositórios escolhidos que realmente foram contribuídos pelo número total de repositórios contribuídos pela primeira vez na amostra. A precisão foi calculada realizando a razão entre repositórios apontados pelo algoritmo que realmente foram contribuídos pelo total de apontamentos do algoritmo.

Concluindo, a metodologia adotada neste trabalho seguiu o modelo CRISP-DM, compreendendo etapas que vão desde a coleta e preparação de dados até a modelagem e avaliação dos algoritmos. O uso de ferramentas como Python e bibliotecas especializadas, aliado a técnicas de aprendizado de máquina, permitiu construir uma abordagem sistemática para categorizar repositórios open source com foco na experiência de desenvolvedores iniciantes. Essa estrutura metodológica não apenas organiza o desenvolvimento do trabalho, mas também assegura a reprodutibilidade e a clareza dos passos realizados.

As seções 5, 6 e 7 detalham a execução do trabalho e a implementação prática do modelo proposto. A seção 5 descreve o entendimento, a aquisição e o pré-processamento dos dados utilizados, abordando aspectos como a seleção de *features* relevantes e o refinamento do dataset. A seção 6 apresenta a modelagem dos algoritmos, especificando os critérios de separação entre os conjuntos de treinamento e teste, além dos ajustes e métricas empregadas. Por fim, a seção 7 discute a análise dos resultados obtidos, comparando o desempenho dos modelos e interpretando os dados em relação dos objetivos do estudo.

5. Compreensão, Aquisição e preparação dos dados

Nesta seção são apresentados os procedimentos referentes a compreensão, aquisição e preparação de dados. A base de dados do GHtorrent[1]. Comprimida ela ocupa aproximadamente 150GB. Após a extração esses dados se expandiram para cerca de 500GB separados em diversos arquivos com a extensão .CSV. Neste trabalho os principais arquivos utilizados foram: *project_member* que possui as informações de membros dos projetos; *commits* que possui as informações dos commits realizados nos projetos; e *projects* que possui as informações de cada projeto.

Com base nos arquivos *project_member, commits* e *projects* foi criado um novo *dataset*, específico para o estudo da integração de novos membros, conforme realizado no trabalho de Liu et al. (2018). No decorrer desta seção estão dispostos em subseções as etapas de pré-processamento, assim como estão presentes as *features* utilizadas e demais processamentos realizados para a construção do *dataset*. As etapas de pré-processamento aplicadas foram: seleção de projetos *open source*; seleção dos *commits* referentes a esses projetos; filtro para primeiras contribuições; o número de membros no momento da

[1]pode ser obtida no endereço: http://ghtorrent-downloads.ewi.tudelft.nl/mysql/, atualizada até 06 de março de 2021

integração; a quantidade de *commits* nesse mesmo período; o intervalo de tempo entre a entrada do desenvolvedor e a criação do projeto; além da primeira e da última integração ocorridas antes da entrada do novo membro.

5.1. Seleção de *features*

Ao estudar as barreiras de contribuição expostas na seção 2 e os dados brutos disponíveis, o estudo selecionou 7 *features*, conforme explorado no trabalho de Liu et al. (2018), para prever a experiência de um iniciante ao integrar-se a um projeto *open source* , incluindo total de *commits*, quantidade de membros, idade do projeto, diferença entre o momento do primeiro *commit* e o momento do ingresso do iniciante, diferença entre o momento do último *commit* e o momento do ingresso do iniciante, diferença entre o momento de ingresso do primeiro membro e o momento do ingresso do iniciante, diferença entre o momento de ingresso do último membro e o momento do ingresso do iniciante. A seguir, discute-se o relacionamento destes *features* com as barreiras de contribuição citadas anteriormente.

A quantidade de *commits* que o projeto possuía no momento da integração também é relevante. Conforme Casalnuovo et al. (2015), um projeto em rápido crescimento costuma apresentar um grande volume de *commits*, necessitando de mais contribuições e, assim, atraindo um maior número de desenvolvedores iniciantes.

Quantidade de membros no momento da integração, projetos com grandes comunidades podem gerar documentação extensa e dispersa, dificultando a navegação. Além disso, a quantidade de membros em um projeto pode influenciar a socialização de iniciantes, indica a probabilidade de haver pessoas experientes disponíveis para oferecer suporte informal. Existem também hipóteses que sugerem que projetos com um grande número de desenvolvedores tendem a ter maior complexidade, o que, por outro lado, pode resultar em um nível mais elevado de organização.

Projetos com idade menor também tendem a ter documentação mais alinhada ao seu estado atual, uma vez que não acumularam mudanças significativas sem atualização correspondente. Projetos mais recentes são frequentemente desenvolvidos com práticas de programação modernas e organizadas, aproveitando tecnologias e padrões atuais, o que contribui para um código mais limpo e simples. A idade recente do projeto pode apontar uma menor complexidade do código porque projetos novos geralmente possuem um escopo inicial mais limitado, com menos funcionalidades e menos linhas de código acumuladas.

O tempo desde o primeiro *commit* reflete a maturidade do projeto, sendo que projetos mais antigos podem acumular documentação excessiva e menos organizada, aumentando a sobrecarga de informações para novos membros.

Tempo desde o último *commit* indica a atividade recente do projeto e, consequentemente, a probabilidade de a documentação estar atualizada. Também assinalam a disponibilidade de tarefas bem definidas e atuais. Além disso, *features* como o número de membros e o tempo desde a primeira integração destacam a abertura do projeto para iniciantes e a probabilidade de suporte para tarefas iniciais.

Diferença entre o momento de ingresso do primeiro membro e o momento do ingresso do iniciante, Em determinados casos, projetos *open source* podem começar como

empreitadas individuais. Assim, a verdadeira data de início de um projeto pode ser considerada o momento da primeira integração de outro desenvolvedor. Em contrapartida, quando um projeto não recebe novas integrações há um longo período, é possível inferir que ele não esteja necessitando nem recebendo novos colaboradores (LIU et al., 2018).

Diferença entre o momento de ingresso do último membro e o momento do ingresso do iniciante. o tempo desde a última integração de membros reflete a receptividade recente do projeto, sugerindo uma maior chance de interação com mentores e membros dispostos a orientar iniciantes.

5.2. Seleção de projetos *open source*

O processo teve início com a análise do arquivo project_members, no qual constam informações sobre os membros dos projetos. A partir dele, foi possível identificar quais projetos são classificados como *open source*. Contudo, um critério adicional foi aplicado a essa seleção: apenas aqueles com mais de 10 membros foram considerados efetivamente *open source*. Esse critério foi adicionado já que projetos menores podem ser apenas repositórios compartilhados e não necessariamente possuir uma comunidade *open source*. Quando o projeto não satisfez esse critério, ele foi excluído da análise. Com essa abordagem, foi possível reduzir o *dataset* inicial de 9.549.783 projetos para 57.075 projetos.

A Figura 4 mostra um histograma da quantidade de projetos pela quantidade de membros. Nele pode-se observar que a curva é muito assimétrica para a direita. O que significa que projetos com muitos membros são menos comuns. Em contrapartida, projetos com poucos membros são extremamente frequentes. Por isso, fez-se necessária a criação do filtro de projetos *open source*.

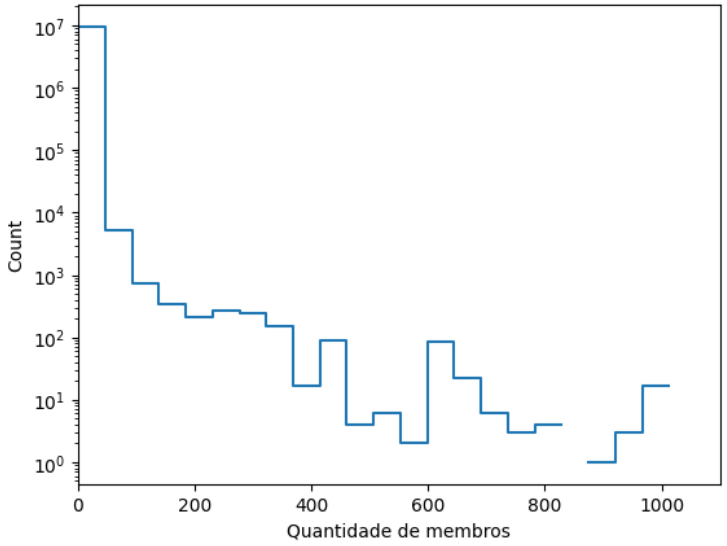

Figura 4. Histograma de membros por projeto. Fonte: Autor

5.3. Seleção de *commits* referentes ao projetos selecionados

Inicialmente o arquivo *commits* possuía a informação de *commits* para todos os projetos. A partir da etapa de seleção de projetos *open source*, que eliminou aqueles que

não atendiam aos critérios estabelecidos, foi realizada a etapa de seleção dos *commits* remanescentes. Esse filtro resultou em um novo arquivo, *open_source_commits*, que contém apenas os *commits* relevantes dos projetos *open source* identificados anteriormente. Esse processo facilita durante o manuseio do *dataset*, tendo em vista que antes do filtro o arquivo era demasiado grande para subir para a memória do computador. De 202GB do arquivo *commits* ele foi fracionado em 410 arquivos que continham somente *commits* de repositórios *open source*, totalizando 2GB de dados.

5.4. Primeira contribuição em projetos *open source*

Um dos objetivos centrais da análise é compreender o processo de integração de novos contribuintes a projetos *open source*. Para isso, nessa etapa de pré-processamento, foi identificada a primeira contribuição de cada membro com base em seu histórico de *commits*, aproveitando a informação de todas as contribuições já realizadas por esse indivíduo em projetos *open source*. A partir da data mais antiga entre esses *commits*, determina-se qual foi sua primeira contribuição, caracterizando o momento em que o desenvolvedor era iniciante na comunidade.

Devido à distribuição dos dados em vários arquivos, o processo foi realizado em duas etapas. Primeiro, filtrou-se a primeira contribuição de cada usuário em cada arquivo. Em seguida, esses resultados foram unificados e novamente analisados para determinar a menor data de contribuição por usuário. Esse procedimento resultou em um total de 162.621 contribuições de iniciantes.

5.5. Primeiro, último e total de *commits* antes da integracão

Após consolidar as informações sobre a primeira contribuição de cada desenvolvedor, o próximo passo consistiu em refinar ainda mais os *commits*, de modo a isolar aqueles *commits* realizados antes da integração de novos colaboradores. Para isso, foi necessário um procedimento que agilizasse a consulta aos arquivos. Sem essa otimização, o código precisaria analisar todos os 400 arquivos de *commits* em projetos *open source* a cada nova integração.

A solução adotada foi criar uma variável do tipo dicionário, na qual os *commits* de cada repositório são armazenados separadamente. Dessa forma, cada integração precisa consultar apenas o arquivo correspondente ao repositório em questão, tornando o processo de cálculo do primeiro e último *commit*, bem como a contagem de *commits*, muito mais eficiente.

Uma vez que os *commits* de cada projeto já estavam separados, somente foi necessário identificar a data mais antiga de um *commit* daquele *dataset*, que não seja do autor original do repositório. Além disso, para calcular o total de *commits* e o último commit foi necessário filtrar aquele *dataset* pela data de ingresso do iniciante. Depois, o registro com a maior data era o último *commit* e a soma de *commits* restantes nesse *dataset* é o total de *commit* antes do ingresso do iniciante.

5.6. Quantidade de membros no momento da integração

Para estimar a quantidade de membros de um projeto no momento da integração de um iniciante, os dados separados no 5.5 anterior (filtro de projetos *open source*) foram analisados considerando o projeto e a data de ingresso de cada iniciante.

Utilizando as datas de ingresso do iniciante e o projeto contribuído, foi possível filtrar somente os ingressos no projeto selecionado anteriores à data de ingresso do iniciante. Após isso, realizou-se a contagem de membros únicos (LIU et al., 2018).

5.7. Primeira e ultima integração antes da adesão do membro iniciante

Uma vez que os dados de integrações de iniciantes estão disponíveis, é possível utilizar a informação de projeto e data de integração do iniciante para obter a informação da primeira e última integração anterior ao membro do registro específico.

Para identificar a primeira integração de um projeto, utilizou-se o arquivo project_members, agrupando as informações por projeto e selecionando a linha com a menor data de entrada que não seja do autor original. De forma semelhante, determinar a data da última integração, efetuou-se o mesmo agrupamento por projeto, buscando a maior data de integração anterior à entrada do novo membro.

5.8. Detalhes dos projetos

No arquivo *projects* estavam dispostas as informações de cada projeto, incluindo URL, nome, autor, data de criação, linguagem de programação, entre outros. Nessa etapa da análise, foram selecionadas certas informações de cada projeto para compor o *dataset* a partir dos projetos filtrados anteriormente nesta seção. Esses dados ajudam a contextualizar as contribuições, bem como o ambiente de desenvolvimento em que o projeto se insere, aprimorando a compreensão do processo de integração dos novos membros.

Os detalhes do projeto provenientes do arquivo *projects* foram utilizados para construir o *dataset* final. Por exemplo, a data de criação do projeto, para determinar a idade do projeto no momento da integração, ou o id do autor, para filtrá-lo fora do *dataset* de membros do projeto.

5.9. Rotulação das instâncias de pessoas experientes

O estudo aborda um problema de classificação binária, mas, até o momento, apenas instâncias de iniciantes estão disponíveis. Dessa forma, foi necessário adicionar amostras de instâncias referentes aos desenvolvedores considerados experientes na comunidade *open source*. Neste trabalho, são consideradas como integrações de desenvolvedores experientes os *commits* realizados em um projeto após a primeira contribuição do indivíduo em qualquer outro projeto *open source*, independentemente do intervalo de tempo entre as contribuições.

Para obter os dados referentes aos desenvolvedores experientes, foram excluídas do arquivo project_members as entradas que coincidiam com o *dataset* de integrações de iniciantes já identificado. Em seguida, aplicou-se, no *dataset* resultante, o mesmo pipeline utilizado para os iniciantes, a fim de determinar os valores das instâncias das integrações de desenvolvedores experientes. A realização desses procedimentos resultou na amostra de instâncias referentes aos desenvolvedores experientes. Essas instâncias foram armazenadas em um arquivo denominado experts_data.

Com todas as informações sobre projetos, membros e *commits* devidamente filtradas e analisadas, um *dataset* final foi construído. Este *dataset* consolida todos os resultados obtidos ao longo do processo, permitindo uma análise completa do envolvimento dos membros e das contribuições dos iniciantes. O Quadro 2 apresenta a estrutura do *dataset*, bem como exemplos de instâncias nele contidas. Dessa forma, o dataset

contém as seguintes features: total_commits, project_members, delta_project_creation, delta_first_onboarding, delta_last_onboarding, delta_first_commit, delta_last_commit, category

No total, o *dataset* resultante do processo possui 162.621 registros referentes à integração de iniciantes e 146.325 registros referentes a ingressos de experientes. Isso, disperso entre 47.418 repositórios e um total de 212.011 desenvolvedores.

total commits	project members	delta project creation	delta first onboarding	delta last onboarding	delta first commit	delta last commit	category
1034	3	1225 days 05:36:43	1225 days 06:36:43	1225 days 06:36:43	1063 days 20:19:08	3 days 02:31:21	beginner friendly
2	23	2093 days 03:58:49	1 days 23:01:12	0 days 01:34:24	673 days 17:04:43	402 days 17:52:45	not beginner friendly

Quadro 2. Exemplo ilustrativo do *dataset* criado para tarefa de classificação abordada neste trabalho

6. Modelagem dos algoritmos

A compreensão, aquisição e preparação dos dados foram fundamentais para possibilitar a construção de um *dataset* consistente e de qualidade. Conforme mencionado na seção 4, sobre a metodologia, alavancando os dados estruturados e as *features* relevantes selecionadas, realizou-se o processo de modelagem dos algoritmos. Que consiste nas etapas de Separação de *dataset* de teste e treinamento, de limpeza e normalização dos dados e de Generalização dos algoritmos, que estão descritas a seguir.

6.1. Limpeza e normalização dos dados

Após as etapas de pré-processamento mencionadas anteriormente neste trabalho, realizou-se a remoção de *outliers* e campos vazios. Para isso, utilizou-se um algoritmo capaz de encontrar todas as linhas em que algum valor fosse nulo e imediatamente removê-las do *dataset*. Na sequência, foi realizada a remoção de dados que iam contra a lógica do *dataset*, como registros em que o ingresso do desenvolvedor acontecera antes da criação do projeto, gerando um *time delta* negativo. Na Figura 5 é possível observar a quantidade de aparições de variações de tempo negativa em relação ao ano do registro. É perceptível que, antes do ano de 2015, isso era muito mais comum de acontecer do que deste ano em diante. Acredita-se que essas situações ocorram pois o repositório teria sido criado primeiro localmente na máquina do autor do projeto e somente após algum tempo o projeto subiu para o Github, uma vez que na época com mais ocorrência desse comportamento o Github não era tão popular quanto nos anos mais recentes.

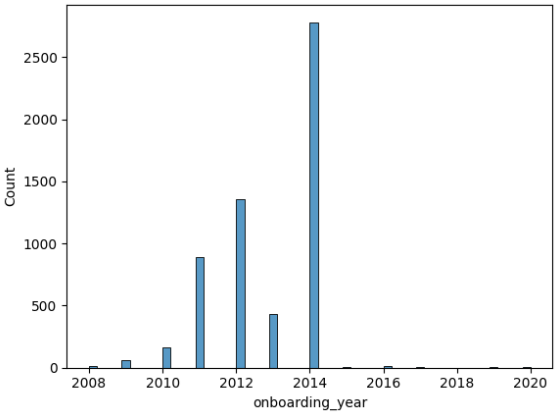

Figura 5. Histograma de *timedeltas* negativos. Fonte: Autor

Depois de limpar os dados, eles precisaram ser normalizados. Para isso, todas as colunas que tinham dados em formato de *time delta* foram convertidas para o total de segundos referente ao *time delta*. Após isso, utilizaram-se os procedimentos de normalização do *Scikit-learn* como a centralização e o escalonamento dos dados.

6.2. Separação entre *dataset* de treinamento e testes

A separação entre dados de treinamento e teste é fundamental para viabilizar o treinamento e possibilitar que a capacidade de generalização do modelo seja avaliada utilizando o *dataset* de teste com dados não vistos durante o treinamento, ou seja, aprender padrões que possam ser aplicados a dados novos e não apenas memorizar as informações apresentadas durante o treinamento.

Afim de manter a integridade temporal da estratégia de divisão entre treino e teste, foram selecionados os dados anteriores a 2017 como treinamento. Dessa forma, apenas dados do "passado"foram utilizados para treinamento. Enquanto os a partir de 2018 foram separados para serem aplicados na etapa de teste, garantindo que apenas dados futuros aos dados de treinamento sejam utilizados para teste. Assim, foi possível simular melhor o comportamento dos dados num ambiente temporal, em que dados do passado são utilizados para prever dados futuros.

6.3. Treinamento dos algoritmos

A etapa de treinamento é a próxima fase do método utilizado. A tarefa de predição de integração de iniciantes foi tratada como um problema de classificação binária. Nesse contexto, o evento de interesse – que consiste na identificação dos projetos indicados para integração de iniciantes – foi considerado como a classe positiva. Assim, o valor "beginner friendly"da variável categoria foi interpretado como a classe positiva na tarefa de predição.

Os algoritmos de Aprendizado de Máquina considerados neste estudo foram: K Nearest Neighbours (KNN), com cinco vizinhos, Suport Vector Machines (SVM) utilizando o *kernel* SBF e Neural Networks (NN) com duas camadas e mil épocas. Conforme mencionado na seção 6.2, os algoritmos foram treinados usando os dados do *dataset* de treino utilizando dados dos anos 2013 à 2017 e avaliados com base no *dataset* de testes dos anos 2018 a 2021.

7. Análise dos resultados

Esta seção corresponde à terceira etapa descrita na metodologia, avaliação do resultado do treinamento. Na sequência são apresentadas as principais características dos conjuntos de dados, juntamente com uma análise descritiva desses dados.

7.1. Análise descritiva dos dados

Esta seção apresenta as principais características observadas nos dados, comparando o comportamento dos valores das *features* entre as classes referentes a integrações de iniciantes e integrações de desenvolvedores experientes em projetos *open source*. A tabela 1 apresenta um compilado dos valores médios para cada *feature*.

Tabela 1. Compilado das médias para cada *feature*

feature	$Iniciantes$	$Experientes$
Total de *commits*	795	350
Total de membros	101	69
Idade do projeto	170 dias	1639 dias
Tempo desde o primeiro membro	105 dias	191 dias
Tempo desde o último membro	15 dias	26 dias
Tempo desde o primeiro *commit*	193 dias	608 dias
Tempo desde o último *commit*	12 dias	325 dias

A *feature* que apresentou a maior discrepância entre integrações de desenvolvedores experientes e iniciantes foi a idade do projeto no momento de integração do iniciante. A média de idade do projeto para integração de iniciantes foi de 170 dias, enquanto para desenvolvedores experientes a média da idade do projeto ficou em 1639 dias. A partir desses dados pode-se inferir que os desenvolvedores iniciantes costumam preferir participar de projetos mais novos. Casalnuovo et al. (2015) afirma que por ser um projeto recente existe uma quantidade maior de tarefas mais simples a serem realizadas, e que o projeto ainda não teve a oportunidade de construir um ecosistema complexo ao seu redor. Essas são algumas hipóteses que sustentam a proporcionalidade inversa entre idade do projeto e atração de iniciantes.

Além do *feature* da idade do projeto, também se observa a diferença do tempo entre o primeiro *commit* do projeto e a integração do usuário iniciante. A média de tempo entre o primeiro *commit* e a integração de um iniciante é 193 dias enquanto para desenvolvedores experientes é 608 dias. Conforme Liu et al. (2018) aponta em seu estudo, algumas vezes o início de um projeto é marcado pelo primeiro *commit* realizado. Assim, é possível identificar que os desenvolvedores iniciantes preferem contribuir em projetos que tiveram seu primeiro *commit* mais recentemente, se comportando de forma muito parecida com a idade do projeto.

Outro *feature* que possui um comportamento interessante é o tempo entre o último *commit* e a integração do iniciante. Para iniciantes, ele apresenta uma média de 12 dias, enquanto para experientes ele fica 325 dias. O que acaba corroborando com as hipóteses levantadas por Liu et al. (2018), que sugere que um projeto frequentemente atualizado pelos desenvolvedores aponta uma insuficiência de pessoal e requer um maior contingente para compartilhamento das tarefas.

Em relação a quantidade total de *commits*, para iniciantes, os projetos apresentam uma média de 795 *commits* enquanto para experientes os dados mostram uma média de 350 *commits* para o projeto no momento de integração do desenvolvedor. A quantidade total de *commits* em um projeto *open source* pode ser um indicador para a integração bem sucedida de um desenvolvedor iniciante. Casalnuovo et al. (2015) aponta que um projeto de crescimento rápido atrai mais desenvolvedores e o seu crescimento pode ser apontado pela quantidade de *commits* realizados no projeto.

Quanto a quantidade total de membros, os projetos apresentaram, em integrações de iniciantes, uma média de 101 membros e para integrações de desenvolvedores experientes a média fica em 69 membros. A quantidade total de membros pode mostrar o tamanho da comunidade em volta de um projeto. Projetos com muitos membros e muita atividade nos *commits* representam uma necessidade maior de novos contribuintes, portanto há mais possibilidades do trabalho disponibilizado por um iniciante seja relevante e integrado ao projeto.

O tempo entre a primeira integração de algum membro e a integração do iniciante apresenta a média de 105 dias para iniciantes e 191 dias para experientes. Liu et al. (2018) aponta que em alguns casos, mesmo que um projeto tenha sido criado há muito tempo pelo seu proprietário, o projeto realmente não começa a funcionar até a participação de novos membros. Apresentando um comportamento parecido com o da idade do projeto.

Quanto ao tempo entre o ingresso do último membro e o momento da integração do indivíduo em questão, apresentou uma média de 15 dias para iniciantes. Em contraste, para experientes a média ficou em 26 dias. Se uma equipe de um projeto permanece estável por muito tempo, significa que este projeto não possui tanta demanda de tarefas. Portanto, não é tao atraente para iniciantes.

7.2. Análise da classificação de acordo com a experiência do desenvolvedor

Por fim, aproximando-se do objetivo mencionado na seção 1. O trabalho investiga se algum dos algoritmos propostos pode recomendar efetivamente os projetos certos para a integração de desenvolvedores iniciantes e compara suas eficácias.

Ressalta-se que, em ordem de evitar piorar as barreiras de contribuição, como a frustração, é mais prudente ter assertividade nas categorizações positivas, mesmo que em detrimento de alguns falsos negativos. Uma vez que o iniciante já entrará no projeto com a expectativa de uma integração mais suave, é pertinente que o algoritmo não aponte repositórios não amigáveis a iniciantes, já que isso aumentaria mais ainda sua frustração.

Tabela 2. Resultados das previsões dos algoritmos

Algoritmo	Acurácia	Precisão	Revocação	Precisão negativa	Revocação negativo
NN	77%	76%	91%	80%	55%
KNN	73%	71%	93%	81%	41%
SVM	66%	64%	97%	82%	17%

A tabela 2 apresenta as medidas de eficácia dos algoritmos. Nela, pode-se observar que o algoritmo que melhor performou foi a algoritmo NN. Este algoritmo atingiu 4 pontos percentuais a mais de acurácia do que o segundo melhor algoritmo, KNN. Porém,

o que a faz se destacar mais do que os outros é a sua revocação negativa. Isso significa que ele apontou menos vezes repositórios não amigáveis a inciantes.

No entanto, a sua revocação negativa, mesmo sendo o melhor dentre os algoritmos, fica próxima de 55%. Ou seja, ele mostra muitos falsos positivos, o que poderia aumentar as dificuldades enfrentadas por iniciantes no seu processo de contribuição ao projeto. O motivo para isso ter acontecido, pode ter sido a seleção desses *features*, que foi realizada focando apenas no comportamento de desenvolvedores iniciantes, ou seja, pode ser que estes *features* não sejam precisos no momento de definir o comportamento de desenvolvedores experientes.

A alta acurácia do algoritmo de *Atificial Neural Network* (ANN) é um sinal de que é possível identificar quando desenvolvedores iniciantes podem ter uma adequada integração em um repositório. Porém, ele ainda apresenta limitações ao isolar os repositórios não amigáveis a iniciantes. Portanto, seria necessário mais estudos explorando *features* que melhor reflitam o comportamento de desenvolvedores experientes, afim de recomendar menos repositórios não amigáveis e atingir um resultado mais satisfatório e assertivo.

Além de avaliar os algoritmos de classificação utilizando todas as *features* disponíveis, também foi realizada a avaliação removendo uma das *features*. Isso foi feito com a finalidade de identificar se os níveis de acurácia obtidos tinham forte dependência de alguma das *features*. Nesse contexto, a ausência de algum *feature* acarretando uma queda demasiada da eficácia do algoritmo de classificação apontaria a existência de forte dependência entre eles.

Tabela 3. Acurácia em função do isolamento de *features*

feature ausente	KNN	SVM	NN
Padrão	73%	66%	72%
Total de *commits*	72%	65%	72%
Total de membros	73%	68%	74%
Idade do projeto	62%	59%	59%
Tempo desde o primeiro membro	74%	67%	75%
Tempo desde o último membro	73%	66%	75%
Tempo desde o primeiro *commit*	66%	59%	61%
Tempo desde o último *commit*	72%	63%	69%

A tabela 3 apresenta a acurácia dos modelos em função do isolamento de *features*. A primeira coluna apresenta as features que foram removidos em cada iteração (exceto pela primeira linha, que apresenta a acurácia do modelo treinado com todos as features). A segunda, terceira e quarta colunas apresentam aos níveis de acurácia atingidos pelos algoritmos KNN, SVM, NN respectivamente. Em geral a maioria dos modelos, obtidos com os algoritmos KNN e NN, apresentou uma acurácia próxima de 70%. Exceto quando removidas as *features* tempo desde o primeiro *commit* e idade do projeto, casos esses em que a acurácia caiu para os três algoritmos. A diminuição da acurácia sugere que as *features* idade do projeto e tempo desde o primeiro *commit* adicionam informações úteis para auxiliar os modelos durante a tarefa de classificação. Tais resultados ajudam a verificar que a acurácia dos modelos não é dependente de uma única *feature*, pois se manteve relativamente estável mesmo na ausência de alguns *features*.

7.3. Limitações e perspectivas futuras

Esta seção discorre a respeito das limitações do trabalho e alguns possíveis caminhos a serem tomados em trabalhos futuros.

Este trabalho procurou investigar quais características de repositórios *open source* podem ser úteis para auxiliar na tarefa de atribuir projetos apropriados para desenvolvedores iniciantes. Portanto, apenas *features* associados aos repositórios foram utilizados. Entretanto, o uso de *features* associados aos usuários podem ser úteis para personalizar e aprimorar a atribuição de repositórios, diminuindo ainda mais a existência das barreiras de contribuição.

O *dataset* utilizado neste trabalho foi o *GHtorrent*, no entanto o seu desenvolvimento parou em 2021. Portanto, dados mais recentes estão faltando no *dataset*. Além disso, dados de anos com desenvolvimento ativo apresentam muitas lacunas. Por exemplo, o ano de 2020 possui poucos registros no *dataset*, enquanto isso o Github apresentou crescimento expressivo em seus dados para o mesmo período.

Para os fins deste estudo, a definição de *open source* aparenta ser um pouco arbitrária. Foi necessário criar um filtro arbitrário para excluir projetos pessoais abertos ou trabalhos de faculdades e manter somente repositórios de *open source* que apresentem relevância na comunidade *open source*. A coleta de dados provenientes de repositórios *open source* é uma demanda para possibilitar que outros trabalhos sejam desenvolvidos. Ou seja, poderia-se buscar filtrar dados dos repositórios *open source* com comunidade ativa, isolando-os efetivamente de demais repositórios que se encaixam na definição de *open source*, mas não possuem uma comunidade em sua volta.

Ao realizar este estudo, encontrou-se dificuldade em utilizar dados anteriores a 2015, uma vez que muitos apresentavam incongruência nas suas informações. Algumas hipóteses foram levantadas para justificar tal característica dos dados. No entanto, somente uma pesquisa mais aprofundada conseguiria responder com exatidão qual seria a razão de acontecer isto com esse *dataset*.

8. Conclusão

Este trabalho buscou compreender e analisar as características de repositórios *open source* que favorecem a integração de desenvolvedores iniciantes, reduzindo barreiras de entrada e fortalecendo as comunidades de software livre.

Os resultados apontaram que desenvolvedores iniciantes tendem a preferir projetos recentes, com alta atividade de *commits* e comunidades maiores, sugerindo que esses fatores reduzem as barreiras de entrada. O uso de redes neurais mostrou-se promissor na recomendação de repositórios adequados, mas ainda existem limitações na capacidade de identificar projetos potencialmente hostis a iniciantes, indicando a necessidade de aprimoramento dos modelos e da inclusão de novas variáveis. O modelo baseado em Redes Neurais apresentou o melhor desempenho, com uma acurácia de 77% na recomendação de repositórios amigáveis para desenvolvedores iniciantes. No entanto, o estudo também identificou limitações, como a dificuldade em isolar repositórios menos receptivos, apontando oportunidades para aprimoramento em futuras pesquisas.

A relevância deste estudo reside na possibilidade de fortalecer comunidades *open source* ao facilitar a integração de novos colaboradores. Entretanto, lacunas como a ausên-

cia de dados mais recentes, limitações na definição de projetos *open source* e dificuldades no isolamento de *features* sugerem caminhos para estudos futuros. Propostas como a personalização das recomendações e o uso de fontes de dados atualizadas podem contribuir significativamente para melhorar a assertividade dos modelos.

Assim, conclui-se que o emprego de aprendizado de máquina é promissor para a promover a inclusão de novos contribuidores em repositórios *open source*. Pelo seu aspecto autônomo e eficiente, é capaz de aumentar a sustentabilidade dos projetos *open source*, como também fomentar inovação e colaboração no ecossistema de software livre.

Referências

AGGARWAL, C. *Data Mining: The Textbook*. Springer International Publishing, 2015. ISBN 9783319141428. Disponível em: <https://books.google.com.br/books?id=cfNICAAAQBAJ>.

BISHOP, C. *Pattern Recognition and Machine Learning*. Springer New York, 2016. (Information Science and Statistics). ISBN 9781493938438. Disponível em: <https://books.google.com.br/books?id=kOXDtAEACAAJ>.

CASALNUOVO, C. et al. Developer onboarding in github: the role of prior social links and language experience. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2015. (ESEC/FSE 2015), p. 817–828. ISBN 9781450336758. Disponível em: <https://doi.org/10.1145/2786805.2786854>.

COSENTINO, V.; IZQUIERDO, J. C.; CABOT, J. Findings from github: Methods, datasets and limitations. In: . [S.l.: s.n.], 2016.

FARIA, M. M. *Detecção de Intrusões em Redes de Computadores com Base nos Algoritmos KNN, K-Means++ e J48*. 2016.

FORD, D.; SHRESTHA, N.; ZIMMERMANN, T. Reboc: Recommending bespoke open source software projects to contributors. In: *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. [s.n.], 2022. p. 1–5. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9833113>.

GÉRON, A. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN 9781491962299. Disponível em: <https://books.google.com.br/books?id=I6qkDAEACAAJ>.

GITHUB. *100 million developers and counting*. 2023. <https://github.blog/2023-01-25-100-million-developers-and-counting/>. Acessado em: 2023-12-02.

GITHUB. *Octoverse: The state of open source and rise of AI in 2023*. 2023. <https://github.blog/2023-11-08-the-state-of-open-source-and-ai/>. Acessado em: 2023-12-02.

GITHUB. *Olá, Mundo*. 2023. <https://docs.github.com/pt/get-started/quickstart/hello-world>. Acessado em: 2023-12-02.

GOUSIOS, G. The ghtorrent dataset and tool suite. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE

Press, 2013. (MSR '13), p. 233–236. ISBN 978-1-4673-2936-1. Disponível em: <http://dl.acm.org/citation.cfm?id=2487085.2487132>.

GOUSIOS, G. The ghtorrent dataset and tool suite. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2013. (MSR '13), p. 233–236. ISBN 978-1-4673-2936-1. Disponível em: <http://dl.acm.org/citation.cfm?id=2487085.2487132>.

JIANG, J. et al. Why and how developers fork what from whom in github. *Empirical Software Engineering*, v. 22, p. 547–578, 2017.

LINOFF, G.; BERRY, M. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. [S.l.: s.n.], 2011. ISBN 9780470650936.

LIU, C. et al. Recommending github projects for developer onboarding. *IEEE Access*, v. 6, p. 52082–52094, 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8458104>.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, JMLR.org, v. 12, n. null, p. 2825–2830, nov 2011. ISSN 1532-4435.

RASCHKA, S. *Python machine learning*. [S.l.]: Packt publishing ltd, 2015.

SARMA, A. et al. Training the future workforce through task curation in an oss ecosystem. In: . New York, NY, USA: Association for Computing Machinery, 2016. (FSE 2016), p. 932–935. ISBN 9781450342186. Disponível em: <https://doi.org/10.1145/2950290.2983984>.

STEINMACHER, I. et al. Social barriers faced by newcomers placing their first contribution in open source software projects. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. New York, NY, USA: Association for Computing Machinery, 2015. (CSCW '15), p. 1379–1392. ISBN 9781450329224. Disponível em: <https://doi.org/10.1145/2675133.2675215>.

STEINMACHER, I. et al. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, v. 59, p. 67–85, 2015. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584914002390>.

VORA, D.; BHATIA, G. *Python Machine Learning Projects: Learn how to build Machine Learning projects from scratch (English Edition)*. Bpb Publications, 2023. ISBN 9789389898262. Disponível em: <https://books.google.com.br/books?id=PXSzEAAAQBAJ>.

WANG, J.; SARMA, A. Which bug should i fix: Helping new developers onboard a new project. In: . New York, NY, USA: Association for Computing Machinery, 2011. (CHASE '11), p. 76–79. ISBN 9781450305761. Disponível em: <https://doi.org/10.1145/1984642.1984661>.

WIRTH, R.; HIPP, J. Crisp-dm: Towards a standard process model for data mining. In: MANCHESTER. *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. [S.l.], 2000. v. 1, p. 29–39.

XIAO, W. et al. Recommending good first issues in github oss projects. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 44.

Proceedings of the 44th International Conference on Software Engineering. Association for Computing Machinery, 2022. p. 1830–1842. Disponível em: <https://dl.acm.org/doi/abs/10.1145/3510003.3510196>.

ZHANG, X. et al. Who will be interested in? a contributor recommendation approach for open source projects. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 29. *The 29th International Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc. and Knowledge Systems Institute, 2017. p. 363–369. Disponível em: <http://ksiresearchorg.ipage.com/seke/Proceedings/seke/SEKE2017_Proceedings.pdf>.