LUIZ EUGÊNIO COELHO NETO

MODELOS PARA ESTIMATIVA DO TEMPO DE RESOLUÇÃO DE ISSUES NO GITHUB UTILIZANDO ATRIBUTOS TEXTUAIS E TEMPORAIS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

Orientador: Giovanni Ventorim Comarela

Coorientador: Gláucia Braga e Silva

VIÇOSA - MINAS GERAIS
2021

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
C672m
2021

Coelho Neto, Luiz Eugênio, 1997-
Modelos para estimativa do tempo de resolução de issues no GitHub utilizando atributos textuais e temporais [recurso eletrônico] / Luiz Eugênio Coelho Neto. – Viçosa, MG, 2021.
1 dissertação eletrônica (64 f.): il.

Orientador: Giovanni Ventorim Comarela.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 61-64.
DOI: https://doi.org/10.47328/ufvbbt.2021.028
Modo de acesso: World Wide Web.

1. Aprendizado do computador. 2. Redes neurais (Computação). 3. GitHub (Programa de computador).
I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação.
II. Título.

CDD 22. ed. 006.31

Bibliotecário(a) responsável: Renata de Fatima Alves CRB6/2578

LUIZ EUGÊNIO COELHO NETO

MODELOS PARA ESTIMATIVA DO TEMPO DE RESOLUÇÃO DE ISSUES NO GITHUB UTILIZANDO ATRIBUTOS TEXTUAIS E TEMPORAIS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 25 de junho de 2021.

Assentimento:

Luiz Eugênio Coelho Neto
Autor

Giovanni Ventorim Comarela
Orientador

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. Além do apoio financeiro, agradeço a todas as pessoas que fizeram parte do meio em que vivi durante o mestrado. Uma vez alguém me ensinou que nossas conquistas são fruto tanto do nosso mérito quanto do nosso meio, meu mérito sozinho não me faria chegar até aqui. Agradeço especialmente aos meus pais, que sempre incentivaram meu estudo desde muito novo. Aos meus orientadores, Giovanni e Gláucia, que formaram uma ótima dupla, onde o conhecimento de um complementava o do outro. Dupla que sempre me ouvia, compreendia e reconhecia minhas qualidades e defeitos. Aos integrantes da república Musa Paradisíaca, Maurício Zacari e Edgard Miranda, que juntos construímos um aconchegante lar, onde sempre tive apoio, compartilhando bons e maus momentos e superando os momentos difíceis de mãos dadas. Esta dissertação marca o encerramento de um maravilhoso ciclo, com muito aprendizado, apoio de amigos e boas lembranças que ficarão marcadas pra sempre e fazem parte da construção de quem eu sou.

Resumo

COELHO NETO, Luiz Eugênio, M.Sc., Universidade Federal de Viçosa, junho de 2021. **Modelos para Estimativa do Tempo de Resolução de Issues no GitHub Utilizando Atributos Textuais e Temporais**. Orientador: Giovanni Ventorim Comarela. Coorientadora: Gláucia Braga e Silva.

Estimar o tempo de resolução de *issues* tem relevância comprovada no contexto dos processos de manutenção de software. No entanto, dentre os modelos de estimativa encontrados na literatura, poucos se referem ao ambiente de *issue tracking* do GitHub. Apesar da popularidade dessa plataforma, especialmente no contexto Open Source, seu sistema de *issue tracking* é pouco burocrático e as *issues* são definidas de forma simplificada, o que torna o processo de construção de modelos preditivos ainda mais desafiador. O objetivo deste trabalho é desenvolver modelos de aprendizado de máquina para estimar o tempo de resolução de *issues* no GitHub, com intuito de auxiliar tarefas como definição de prazos. A fim de suprir a escassez de dados, são propostos atributos textuais, responsáveis por capturar características sobre o texto das *issues*; e temporais, responsáveis por fornecer informações sobre o momento em que ocorreram eventos relacionados a elas. Redes neurais (MLP) também são aplicadas para classificação e provam serem mais adequadas para resolução do problema. Para fins de validação, os modelos propostos são comparados com uma referência encontrada na literatura, revelando resultados positivos por meio de diversas métricas, destacando-se uma melhora significativa na acurácia. Por fim, são realizados testes após a adição das categorias de atributos propostas (textuais e temporais) a fim de avaliar o impacto causado por elas na qualidade das estimativas.

Palavras-chave: Rastreamento de problemas. Estimativa do tempo de resolução. Aprendizado de máquina. Redes neurais. GitHub.

Abstract

COELHO NETO, Luiz Eugênio, M.Sc., Universidade Federal de Viçosa, June, 2021. **Models for Estimating Issue Resolution Time on GitHub Using Textual and Temporal Features**. Advisor: Giovanni Ventorim Comarela. Co-advisor: Gláucia Braga e Silva.

Estimating issues resolution time is one of the most important step in software maintenance processes. However, although the subject is covered in the literature, there are few specific models for GitHub. This platform is very popular mainly in the open source context but its issue tracking system is not bureaucratic and issues are registered in a very simple way, which makes the process of building predictive models even more challenging. This work aims to develop machine learning models to estimate the resolution time of *issues* from GitHub. To handle with the data scarcity, we propose textual attributes to capture issues characteristics; and temporal attributes to provide information about the time of issue events. Neural networks were used in classification algorithms and proved to be more suitable for solving this problem. To validade the proposed models we compared them with a reference from literature through different metrics and the results were positive with a significant improvement in accuracy. Finally, to validate the new attributes we conduct comparative tests in order to assess the impact of their inclusion on the estimations quality.

Keywords: Issue tracking. Issue lifetime prediction. Machine Learning Model. Neural networks GitHub.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Sistemas de *issue tracking* são comumente utilizados por equipes de desenvolvimento de software distribuídas, assumindo um papel indispensável no controle das mudanças que ocorrem nos projetos. Nesses sistemas, são reportadas questões a serem resolvidas sobre o software (*issues*), tais como solicitações de correção de erros, sugestões de melhorias e até mesmo novos requisitos. Uma vez reportada, a *issue* pode ser atribuída/assumida por um colaborador (desenvolvedor) e os envolvidos iniciam uma discussão em torno dela, postando comentários no ambiente (FAN, 2010; GIGER; PINZGER; GALL, 2010). Em projetos Open Source (OSS), qualquer um pode colaborar e reportar *issues*, seja um desenvolvedor da equipe, um colaborador ou usuário final (CROWSTON et al., 2006). Após os colaboradores solucionarem a questão, a *issue* é fechada e o tempo desde que ela foi aberta até seu fechamento é chamado de tempo de resolução.

Estimar o tempo de resolução de *issues* pode ser considerado um tipo de estimativa de esforço (AL-ZUBAIDI et al., 2017; ANBALAGAN; VOUK, 2009), mas com foco em pequenas partes do software. O propósito da estimativa de esforço é prever a quantidade de esforço requerido para completar o conjunto de tarefas que compõem o ciclo de vida do projeto (MENDES, 2014). Nesse caso em específico, o esforço é considerado em termos do tempo demandado e a automação das estimativas mostra-se relevante para atender diferentes necessidades dos atores em projetos OSS. As estimativas são úteis para interessados em gerenciar os projetos, ao definir prazos para tarefas que dependem da resolução de determinadas *issues* ou ao direcionar recursos, por exemplo, investindo mais esforço em *issues* previstas com longa duração. Já para os colaboradores, considerando que o primeiro passo em comum é escolher qual *issue* trabalhar (KALLIAMVAKOU et al., 2015), a estimativa fornece uma referência para o tempo demandado e pode auxiliar no processo de seleção. Do ponto de vista dos usuários finais, que reportaram *issues* no projeto, a estimativa fornece um *feedback* rápido de quando o(s) problema(s) reportado(s) será(ão) resolvido(s).

Apesar de diversos modelos para estimativa do tempo de resolução de *issues* serem encontrados na literatura, poucos são desenvolvidos para o GitHub (GIGER; PINZGER; GALL, 2010; ARDIMENTO; DINAPOLI, 2017), um dos repositórios de projetos OSS mais utilizados atualmente. Essa plataforma oferece diversos recursos para o gerenciamento da configuração de um projeto de software, integrando funcionalidades como versionamento, *pull requests* e *issue tracking* (GITHUBGUIDE, 2020). Embora o uso da ferramenta de *issue tracking* seja opcional no GitHub, seu uso atrai mais colaboradores e está presente nos projetos mais populares (BISSYANDÉ et al., 2013), por exemplo,

a *Scikit-learn*[1], *Pandas*[2], *Numpy*[3] e *Keras*[4], cujos softwares estão sendo utilizados nesta pesquisa.

1.1 Problema e Hipóteses da Pesquisa

Considerando-se a relevância das estimativas de esforço no contexto de projetos *Open Source* e diante das lacunas observadas na literatura quanto a modelos de estimativa direcionados ao GitHub, as seguintes hipóteses serão investigadas nesse trabalho:

1. H1: É possível estimar com acurácia e precisão o tempo de resolução de *issues* no GitHub, considerando-se a não estruturação das informações sobre elas.
2. H2: A adição de atributos textuais resulta em melhorias na qualidade das estimativas.
3. H3: A adição de atributos temporais resulta em melhorias na qualidade das estimativas.
4. H4: O uso de algoritmos de aprendizado, como redes neurais, mostra-se mais adequado para resolver o problema.

Apesar da popularidade do GitHub, existem poucos estudos voltados para estimativa do tempo de resolução de *issues* na plataforma, por essa razão a hipótese H1 será abordada. Ao se reportar uma *issue* no GitHub, a única informação obrigatória é o título. Além disso, o sistema é menos burocrático, com *issues* postadas sem revisão e aprovação da equipe. Dessa forma, dificulta-se a construção de modelos preditivos com base apenas em dados brutos e faz-se necessário a extração de informações relacionadas às *issues*. Grande parte dos atributos propostos neste trabalho pertencem às categorias dos atributos textuais, que são informações extraídas dos textos presentes nas *issues*; e temporais, que consistem em informações sobre o momento em que ocorreram eventos relacionados às *issues*. É importante avaliar se a adição desses novos atributos traz melhorias para as estimativas realizadas e por essa razão as hipóteses H2 e H3 são investigadas. *Issues* postadas sem revisão podem não ser relevantes e nunca serem respondidas, se comportando como ruído. Por essa razão, é importante a investigação da hipótese H4 e o uso de algoritmos de aprendizado menos sensíveis a ruído como é o caso das redes neurais artificiais.

[1] https://github.com/scikit-learn/scikit-learn
[2] https://github.com/pandas-dev/pandas
[3] https://github.com/numpy/numpy
[4] https://github.com/keras-team/keras

1.2 Objetivo

Diante da relevância de se estimar o tempo de resolução de *issues* em projetos de software e considerando as lacunas existentes na literatura sobre esse tipo de estimativa no contexto do GitHub, o objetivo deste trabalho é desenvolver modelos de aprendizado de máquina, utilizando atributos textuais e temporais, para prever o tempo de resolução de *issues* nesta plataforma. A fim de atingir o objetivo principal e responder às questões de pesquisa, os seguintes objetivos específicos são considerados:

1. Desenvolver modelos de referência a partir do trabalho de Kikas, Dumas e Pfahl (2016).
2. Propor novos atributos com informações relacionadas às *issues*, a fim de aprimorar os modelos de referência.
3. Aprimorar os modelos de referência, a partir do uso de algoritmos de aprendizado mais eficazes, como redes neurais.
4. Avaliar a importância dos atributos ao realizar as estimativas.
5. Avaliar o desempenho dos novos modelos, comparando-os com os modelos de referência e identificando as melhorias.
6. Disponibilizar os dados e códigos necessários para replicação e aprimoramento da pesquisa no GitHub.

1.3 Relevância da Pesquisa

Além do valor das estimativas para os atores envolvidos em processos de desenvolvimento de software no GitHub descrito na introdução, a investigação das hipóteses e realização dos objetivos também é relevante sobre diversos aspectos. O desenvolvimento de novos atributos fornece meios de extrair informações relacionadas às *issues* que podem ser reaproveitados no contexto de outros trabalhos. Os atributos propostos podem ser utilizados tanto no desenvolvimento de outros modelos para estimar o tempo de resolução ou para outros fins, como a análise e exploração de dados em estudos empíricos. Esses atributos também podem ser utilizados em trabalhos com dados de outras plataformas, além do GitHub, uma vez que existem informações comuns entre todos os sistemas de *issue tracking*. Além dos novos atributos, os modelos propostos também podem ser úteis em outras pesquisas, como em estudos sobre priorização de *issues* e *pull requests*, onde a estimativa do tempo de resolução é considerada uma informação relevante (DHASADE; VENIGALLA; CHIMALAKONDA, 2020). Por fim, os conhecimentos obtidos no desenvolvimento dos modelos como o impacto da aplicação das redes neurais agregam para o estado da arte e podem ser aplicados em trabalhos futuros.

1.4 Organização da dissertação

O trabalho está organizado da seguinte forma: o Capítulo 2 explica os principais conceitos presentes neste trabalho, desde o funcionamento e organização do GitHub, até definições sobre algoritmos de aprendizado de máquina. O Capítulo 3 apresenta os principais estudos sobre estimativa do tempo de resolução de *issues* e trabalhos voltados para análise de dados no GitHub. A metodologia utilizada neste trabalho é descrita no Capítulo 4. A aquisição e a preparação dos dados necessários é relatada no Capítulo 5. O Capítulo 6 descreve o desenvolvimento dos modelos de referência, apresenta os atributos propostos, e os passos para o treinamento dos novos modelos. Os resultados são discutidos no Capítulo 7, junto com algumas limitações da pesquisa. Por fim, o Capítulo 8 traz a conclusão e alguns trabalhos futuros.

2 Referencial Teórico

Este capítulo apresenta as principais características do desenvolvimento de software em projetos *open source* no GitHub, destacando as particularidades do seu sistema de *issue tracking*, quais são os atores do processo e como eles se organizam nos projetos. Na sequência, o Capítulo aborda alguns fundamentos sobre estimativa de esforço nesse contexto particular e destaca as principais características das técnicas orientadas a aprendizado de máquina aplicadas na pesquisa.

2.1 Desenvolvimento de software no GitHub

O GitHub é um plataforma amplamente utilizado para hospedagem de projetos *open Source*, seja por empresas privadas, comunidades *open source* ou uso pessoal. Nele, são hospedados diversos projetos importantes, tais como softwares da Google e Microsoft, bibliotecas como Sklearn e Keras, entre outros. O GitHub é construído com base no Git, um sistema de controle de versão responsável por gerenciar alterações nos softwares. As alterações são efetivadas por meio de *commits*, sendo possível a revisão colaborativa do código (KALLIAMVAKOU et al., 2014). Além da hospedagem e versionamento de código, a plataforma também fornece um aparato de ferramentas integradas a fim de oferecer suporte para o desenvolvimento de software em equipe, dentre elas o sistema de *issue tracking*.

2.1.1 O sistema de Issue tracking do GitHub

Sistemas de *issue tracking* ou sistemas de rastreamento de questões auxiliam equipes no controle e rastreamento das mudanças que ocorrem no projeto. Nesses sistemas, são relatadas questões como erros no código ou requisições de melhorias no software. Os relatórios das questões identificadas são armazenados como *issues* e o sistema torna viável o gerenciamento delas, sendo possível revisar as *issues* em aberto, atribuí-las para alguém ou marcá-las como resolvidas. Um histórico das *issues* resolvidas também é armazenado, fornecendo informações sobre o trabalho já realizado no projeto (BERTRAM et al., 2010). Embora haja variações nas diversas ferramentas de *issue tracking*, em geral, uma *issue* é caracterizada pelos seguintes atributos (GITHUBGUIDE, 2020; BUGZILLA, 2018):

- Título;
- Descrição do problema/questão;
- Usuário atribuído para trabalhar na resolução da *issue*;

- Milestone: etapa do processo ou recurso específico em que a issue faz parte;
- Labels: palavras chave que ajudam a categorizar ou filtrar as *issues*;

Após reportada, uma *issue* possui o status "aberta". Os *stakeholders* discutem sobre o problema e suas possíveis soluções postando comentários em torno de uma discussão, de forma similar a um fórum. Quando a questão é dada como resolvida algum *stakeholder* altera o status da *issue* para "fechada". O tempo desde a abertura da *issue* até o fechamento é chamado de tempo de resolução, *life-time*, ou *fix-time* (BERTRAM et al., 2010).

No GitHub, algumas particularidades tornam o sistema de *issue tracking* menos burocrático e incentivam a participação de usuários finais. Primeiramente, qualquer pessoa pode reportar ou comentar em uma *issue* e o único campo obrigatórios é o título. Além disso, não existe um workflow bem definido para a *issue*. Ao contrário do que ocorre em ferramentas como o Bugzilla[1], onde uma *issue*/*bug* primeiramente precisa ser revisada para depois ser aprovada.

Como consequência da falta de burocracia, poucas informações sobre as *issues* são geradas no momento da criação, além do conteúdo textual. No entanto, como no GitHub o sistema de *issue tracking* é integrado ao controle de versão, podem existir informações adicionais sobre os eventos ocorridos no projeto. Os eventos relacionados a uma *issue* ocorrem após sua criação e podem ser dos seguintes tipos (GITHUB, 2020):

- **Fechamento**: quando a *issue* é dada como resolvida;
- **Reabertura**: quando a *issue* dada como resolvida e fechada anteriormente, é aberta novamente;
- **Referenciação**: quando a *issue* é mencionada em uma mensagem de *commit*;
- **Menção**: quando a *issue* é mencionada em outra *issue* ou *pull request*, no título, descrição ou comentários;
- **subscrição**: quando um usuário ativa as notificações de atualizações na *issue*;

Esses eventos ocorrem no decorrer do desenvolvimento, como consequência do fluxo de informações ao se realizar uma contribuição em um projeto. A integração dos sistemas no GitHub fornecem uma grande quantidade de informações sobre o projeto em geral, especificamente relacionadas com as *issues* devido a sua importância no fluxo da contribuição. Os passos presentes nesse fluxo serão detalhados na próxima seção.

[1] https://www.bugzilla.org/docs/4.2/en/html/lifecycle.html

2.1.2 Fluxo da contribuição

O GitHub introduziu o desenvolvimento mediante etapas de "*fork & pull*", onde os desenvolvedores criam suas próprias cópias de projetos (forks), e submetem *pull requests* quando desejam que suas alterações sejam integradas ao projeto principal (KALLIAMVAKOU et al., 2014). De acordo com a pesquisa de Kalliamvakou et al. (2015), os desenvolvedores tendem a seguir um mesmo conjunto de passos ao realizarem uma contribuição. Primeiramente, escolhe-se uma *issue* a ser trabalhada, e cria-se uma cópia do projeto a fim de realizar *commits* com modificações sem interferir no projeto principal. Posteriormente, é submetida uma *pull request*, apresentando o código desenvolvido como possível solução para a questão relatada na *issue*. Por fim, membros do grupo principal de desenvolvedores revisam o código e, mesclam ao código principal caso seja aprovado, ou solicitam mais alterações, quando necessário. Dessa forma, espera-se que a contribuição do desenvolvedor seja efetivada, considerando a *pull request* e a *issue* relacionada como resolvidas.

Diversos atores participam desse fluxo da contribuição, uma vez que não há regras bem definidas para os papéis dos *stakeholders* em projetos *open source*. Isso ocorre pois qualquer pessoa tem a liberdade de executar, copiar, distribuir, estudar, alterar e melhorar o software. Contudo, a equipe tende a ser organizada em camadas, tanto em projetos hospedados no GitHub quanto em outras plataformas.

2.1.3 Atores do processo no contexto open source

Embora a organização e gerência do projeto varie de comunidade para comunidade, alguns estudos na literatura indicam que as equipes normalmente seguem uma estrutura hierárquica, com o formato similar ao de uma cebola (Gacek; Arief, 2004; MOON; SPROULL, 2000; JOBLIN et al., 2017). No centro, estão os desenvolvedores do core, sendo os que mais contribuem com o projeto. O *core* é o menor grupo, cerca de 20% dos desenvolvedores que mais contribuem e são responsáveis por 80% das contribuições (CROWSTON et al., 2006; MOCKUS; FIELDING; HERBSLEB, 2002). Tal grupo toma decisões de gerenciamento do projeto, modificam o código e revisam códigos de outros desenvolvedores. Eles também desempenham um papel central na comunicação e liderança e costumam ter um forte conhecimento sobre a arquitetura do sistema (CATALDO; HERBSLEB, 2008; TERCEIRO; RIOS; CHAVEZ, 2010). Em torno do *core*, encontram-se os co-desevolvedores, que estão envolvidos na identificação de *bugs* ou melhorias (*issues*), discussões sobre propostas de soluções e participam, esporadicamente, do desenvolvimento. Em algum momento eles submetem códigos, que, por sua vez, são revisados pelo *core*. Por fim, encontram-se os usuários ativos e passivos. Usuários ativos não participam do desenvolvimento, mas reportam *issues*, contribuem com a documentação e fornecem casos de uso, à medida que testam o software. Já os usuários passivos somente

utilizam o software e não realizam contribuições (Gacek; Arief, 2004; CROWSTON et al., 2006).

Os papéis em cada grupo são dinâmicos, bem como os participantes e suas respectivas tarefas. Co-desenvolvedores podem se tornar membros do *core* à medida que fazem contribuições relevantes e ganham reputação. O inverso também ocorre, quando um desenvolvedor do *core* começa a se envolver menos e passa a ser um co-desenvolvedor, podendo ainda transitar para o grupo de usuários ativos ou passivos (Gacek; Arief, 2004).

À medida que esses atores transitam entre os grupos e ao decorrer do fluxo da contribuição, a ”memória” do trabalho da equipe vai se construindo e sendo arquivada nos sistemas. Os dados presentes na plataforma do GitHub contem informações valiosas sobre produtividade da equipe, rastreabilidade de mudanças, retrabalho, dentre outras. Com base nas características de cada grupo podem ser extraídas informações sobre os envolvidos no projeto. Sendo assim, dados sobre o projeto como um todo podem ser analisados e aproveitados para realização de estimativas sobre as *issues*.

2.2 Estimativa de esforço em projetos de software

Estimar o esforço é uma tarefa crítica em projetos de software, pois abrange muitas finalidades, tais como, estimativa de custo, alocação de recursos, cronograma e orçamento (DÔRES, 2015). O propósito da estimativa de esforço é prever a quantidade de esforço requerido para completar o conjunto de tarefas que compõem o ciclo de vida do projeto. Abordagens existentes, em geral, mensuram o esforço por meio da métrica pessoa-hora. As estimativas são baseadas em entradas como, dados de projetos similares ou características relacionadas ao esforço, dentre elas, medidas do tamanho do sistema, complexidade, fatores de produtividade e quantidade de tarefas de manutenção (MENDES, 2014). As técnicas de estimativa de esforço são categorizadas, segundo a taxonomia proposta por (BOEHM; ABTS; CHULANI, 2000), em:

- **Baseadas em modelo**: englobam as técnicas tradicionais, que utilizam equações fixas, definidas com base em dados de projetos de software anteriores.
- **Baseadas em expertise**: englobam técnicas que se baseiam na análise de especialistas.
- **Dinâmicas**: englobam técnicas que consideram as mudanças nas estimativas, de forma dinâmica, com o passar do tempo da realização do projeto;
- **Baseadas em regressão**: estimam o esforço como uma variável dependente e usa o tamanho de software como uma variável independente.

- **Compostas**: fazem o uso de uma combinação de técnicas para se obter melhor acurácia nos resultados.
- **Baseadas em Aprendizado de Máquina**: técnicas que fazem o uso de algoritmos de aprendizado de máquina.

Diante das dificuldades de aplicação das técnicas tradicionais no início de um projeto, tais como o COCOMO (BOEHM et al., 1995), por não haver informações e conhecimentos suficientes, técnicas baseadas em aprendizado de máquina se apresentam como uma alternativa promissora. A criação de modelos de estimativa desse tipo ocorre a partir da exploração de dados históricos, feita por algoritmos específicos, a fim de inferir um conjunto de regras que permitam a dedução de valores futuros (SRINIVASAN; FISHER, 1995). Dentre as abordagens de aprendizagem de máquina para modelos preditivos de esforço destacam-se: Redes Neurais Artificiais, Árvores de Classificação e Regressão e Algoritmos Genéticos. Uma vez que é complexa a natureza do problema de estimar o esforço, a solução por abordagens matemáticas clássicas é limitada, pois utiliza equações fixas em projetos diferentes. Nesse contexto, as redes neurais artificiais são mais eficientes, pois sua capacidade de adaptação, aprendizado, e tolerância a ruídos as tornam adequadas para problemas desse tipo (DÔRES, 2015).

Métodos baseados em aprendizado de máquina consistem em uma solução mais apropriada para contextos com características particulares, como o desenvolvimento de projetos de software open source (OSS). Nesse contexto, métodos tradicionais de estimativas de esforço, como o COCOMO, não são adequados (FERNANDEZ-RAMIL; IZQUIERDO-CORTAZAR; MENS, 2009) por causa da natureza distribuída e colaborativa do desenvolvimento. O esforço investido normalmente não é conhecido, mesmo após a conclusão do projeto. Empresas individuais normalmente sabem do esforço que investiram, mas não possuem clareza sobre o esforço de outras comunidades e indivíduos que colaboraram com o projeto. Entretanto, o interesse nessas informações tem aumentado, especialmente por causa do crescimento da quantidade de empresas em que a iniciativa OSS se tornou relevante para sua estratégia de negócio (ROBLES et al., 2014).

No universo do desenvolvimento OSS, mudanças ocorrem a todo momento, em especial pela natureza colaborativa dos projetos. Dessa forma, uma demanda que se apresenta é a estimativa do esforço para aplicar tais mudanças, por exemplo, o esforço em termos do tempo gasto na resolução das *issues* reportadas. Assim, estimar o tempo de resolução de *issues* tem como foco pequenas partes do software e pode ser considerado um tipo de estimativa de esforço (AL-ZUBAIDI et al., 2017; ANBALAGAN; VOUK, 2009). Contudo, esse tipo de estimativa difere dos métodos tradicionais, onde se considera o esforço gasto no projeto como um todo. Tal diferença de objetivos é condizente com o desenvolvimento de software moderno, utilizando metodologias iterativas e incrementais, onde partes do software são desenvolvidas a cada iteração (AL-ZUBAIDI et al., 2017).

2.3 Algoritmos de Aprendizado de Máquina

O campo de aprendizado de máquina ou *machine learning* é focado no desenvolvimento de algoritmos capazes de se aprimorarem com a experiência, sem serem explicitamente programados. Tais algoritmos são responsáveis por aprender e reconhecer, de forma automatizada, padrões complexos sobre os dados, e utilizar esses padrões para tomar decisões inteligentes como, estimativa de variáveis, ou agrupamento dos dados (MITCHELL, 1997; HAN; KAMBER; PEI, 2012). Algoritmos de aprendizado de máquina automatizam a construção dos modelos analíticos para a realização das estimativas descritas neste trabalho. O funcionamento dos dois algoritmos utilizados será brevemente descrito nas próximas subseções.

2.3.1 Random Forest

Random Forest ou florestas aleatórias é um método *ensemble* para classificação ou regressão. Métodos *ensemble* utilizam um modelo composto, construído a partir da combinação de vários modelos individuais. Dada uma nova instância de dados, cada modelo individual ”vota” realizando uma previsão e a saída final é a classe mais votada (classificação) ou a média das previsões (regressão). De acordo com Han, Kamber e Pei (2012), o treinamento dos modelos ocorre realizando amostragens do conjunto de dados, sendo que para cada um dos k modelos de aprendizado ($M1,M2,...,MK$) é criado um subconjunto de instâncias selecionadas aleatoriamente, com repetição. Sendo assim, dado um conjunto de dados D, o subconjunto aleatório Di é utilizado para treinar o modelo Mi, com i variando de 1 até k, onde k é a quantidade de modelos individuais.

No caso das *Random Forests*, cada modelo Mi no conjunto é uma árvore de decisão (*decision tree*), e a coleção de árvores é chamada de floresta. As árvores individuais são geradas por meio de diversos conjuntos de características/atributos dos dados, selecionados aleatoriamente para cada nó. Em cada nó, esse conjunto aleatório de atributos é avaliado, e o atributo que melhor divide os dados é escolhido (HAN; KAMBER; PEI, 2012). Em resumo, *random forest* é um modelo composto por várias árvores de decisão, treinadas com diferentes partes do conjunto de dados, considerando somente um subconjunto de atributos ao escolher o atributo de cada nó.

2.3.2 Multilayer perceptron

Multilayer perceptron é um tipo de rede neural artificial (ANN) feed foward, composta por múltiplas camadas de perceptrons. Em suma, redes neurais artificiais são um conjunto de nós com entrada e saída conectadas, onde cada conexão possui um peso associado. No processo de aprendizado, os pesos são iterativamente ajustados a fim de prever corretamente os rótulos das instâncias de dados. Redes Multilayer perceptron possuem

3 ou mais camadas de nós, sendo 1 camada de entrada, 1 ou mais camadas ocultas e 1 camada de saída. Os nós da camada de entrada são alimentados pelos atributos dos dados. Tal camada pondera os atributos e alimenta a segunda camada (camada oculta), que por sua vez, pondera e alimenta a próxima. Por fim, as saídas da última camada oculta alimentam a camada de saída, responsável por emitir a previsão para as instâncias dos dados (HAN; KAMBER; PEI, 2012).

Os nós da rede são compostos por perceptrons e calculam a saída com base em uma combinação linear dos valores de entrada, ponderando esses valores com os pesos (MITCHELL, 1997). Uma vez que as camadas são totalmente conectadas, cada perceptron de uma camada se conecta com todos os perceptrons da camada seguinte. A rede é *feed forward* pois nenhuma das conexões voltam para o mesmo perceptron ou para a camada anterior. O aprendizado ocorre através do ajuste iterativo do conjunto de pesos. Após cada parte dos dados ser processada, os rótulos previstos são comparados com o resultado esperado e pesos são ajustados a fim de minimizar o erro (HAN; KAMBER; PEI, 2012).

2.4 Métricas para avaliação de modelos de *Machine Learning* (ML)

Ao desenvolver um modelo de aprendizado de máquina, torna-se importante mensurar quão corretas são as estimativas realizadas. Por essa razão, diversas métricas disponíveis na literatura avaliam modelos de diferentes formas. Tais métricas também possibilitam a comparação de modelos e métodos distintos, a fim de determinar o mais adequado. As métricas comumente utilizadas na literatura e que também são aplicadas neste trabalho serão definidas nesta seção, com base no livro de Han, Kamber e Pei (2012).

Para compreensão das equações definidas nas próximas subseções torna-se fundamental o entendimento de alguns termos. Considerando-se um problema de classificação binária, por exemplo, definir se uma *issue* será resolvida ou não. As instâncias positivas (P) possuem como rótulo a principal classe de interesse e as negativas (N) a outra classe. No caso do exemplo, as positivas seriam *issues* que foram resolvidas, e as negativas as que não foram. Com base nesses conceitos, os seguintes termos são definidos:

- **True Positives** (TP): Instâncias da classe positiva previstas corretamente. TP é a quantidade de verdadeiros positivos;
- **False positives** (FP): Instâncias da classe negativa incorretamente previstas como positivas. FP é a quantidade de falsos positivos;
- **True negatives** (TN): Instâncias da classe negativa previstas corretamente. TN é a quantidade de verdadeiros negativos;
- **False negatives** (FN): Instâncias da classe positiva previstas incorretamente como negativas. FN é a quantidade de falsos negativos;

2.4.1 Acurácia

Dado um conjunto de dados, a acurácia é a proporção de instâncias desse conjunto classificadas corretamente. Ela reflete o quão bem o classificador reconhece as instâncias das várias classes. A fórmula da acurácia é apresentada na equação 2.1.

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

2.4.2 Precisão e revocação

A exatidão das previsões realizadas é medida pela precisão. Tal métrica calcula a proporção de instâncias previstas como positivas que realmente eram positivas. Levando em consideração o exemplo em questão, a precisão seria a proporção das *issues* previstas como resolvidas, que realmente foram resolvidas. A equação 2.2 define a fórmula da precisão.

$$\text{Precisão} = \frac{TP}{TP + FP} \tag{2.2}$$

A revocação mede a integridade ou sensibilidade das previsões, por meio da proporção das instâncias positivas, previstas como positivas. Para o exemplo a revocação seria, a proporção de *issues* resolvidas que foram previstas. A revocação é definida pela fórmula presente na equação 2.3

$$\text{Revocação} = \frac{TP}{TP + FN} \tag{2.3}$$

A precisão e a revocação definidas nesta seção referem-se somente à classe positiva. Entretanto, tais métricas podem ser igualmente calculadas para a classe negativa, ou para qualquer uma das classes em um problema multi-classe (não binário). Uma precisão perfeita significa que todas as instâncias previstas como uma determinada classe X, realmente são da classe X. Contudo, isso não diz nada sobre as instâncias dessa classe X que não foram previstas, sendo essa possibilidade mensurada pela revocação. Uma revocação perfeita significa que todas as instâncias de uma classe X foram previstas. Geralmente, existe uma relação inversa entre precisão e revocação, podendo uma ser melhorada ao custo de deteriorar a outra. Por exemplo, o classificador pode obter uma alta precisão prevendo que uma *issue* será resolvida somente quando tiver muita certeza, ao custo de uma baixa revocação, pois haverá muitas *issues* que foram resolvidas e não foram previstas.

2.4.3 F-score

Uma forma de avaliar tanto a precisão quanto a revocação é combinando as duas em uma única métrica. O F-score é a média harmônica da precisão e revocação, sendo necessário um bom valor em ambas (acima de 0.5) para um classificador obter um bom F-score. O cálculo do F-score é realizado de acordo com a equação 2.4:

$$F - score = \frac{2 \times \text{precisão} \times \text{revocação}}{\text{precisão} + \text{revocação}} \tag{2.4}$$

2.5 Mineração de Texto

Mineração de textos consiste em processos para extração de informações a partir de textos, onde diversos algoritmos e métodos de aprendizado de máquina podem ser aplicados com o objetivo de encontrar padrões úteis no texto (NAHM; MOONEY, 2002). Tais processos são relevantes para modelos de previsão como uma forma de extração de atributos sobre os dados.

Uma vez que textos geralmente consistem em dados não estruturados, a etapa inicial do processo de mineração é a representação do texto em uma estrutura de dados. Dentre as estruturas mais utilizadas tem-se: a *bag of words*, onde cada documento textual é representado por um vetor de palavras; e as matrizes TF-IDF, onde o documento é representado por um vetor de frequências ponderadas dos termos. Também pode ser incluída uma etapa para pre-processamento executando tratamentos no texto como, remoção de pontuação e *stopwords*, palavras que não agregam significado. Por fim, a partir da representação estruturada, são aplicados métodos para análise do texto, dentre eles, extração de informação, processamento de linguagem natural, sumarização, métodos de aprendizagem não supervisionada (*i.e.* agrupamento, modelagem de tópicos) (ALLAHYARI et al., 2017; SRIVASTAVA; SAHAMI, 2009).

Modelagem de tópicos é uma técnica bastante popular de agrupamento probabilístico de textos. O objetivo principal é criar um modelo generativo para o conjunto de documentos, considerando os documentos como uma mistura de tópicos e os tópicos como uma distribuição probabilística de palavras. Latent Dirichlet Allocation (LDA) é um modelo considerado o estado da arte para extração de informações temáticas (tópicos) de uma coleção de documentos (ALLAHYARI et al., 2017; BLEI; NG; JORDAN, 2003). A ideia por trás do LDA é modelar o documento como uma combinação de tópicos latentes, onde cada tópico é definido como uma distribuição da probabilidade de ocorrência de cada termo. De forma geral, assume-se que N tópicos são associados a um conjunto de documentos e estão presentes em diferentes proporções em cada um deles. O desafio do LDA consiste em aprender quais são esses tópicos com base nos dados, de forma não supervisionada (SRIVASTAVA; SAHAMI, 2009). Uma vez calculada a distribuição

dos tópicos nos documentos, o LDA pode ser usado como um módulo em modelos mais complexos e com diferentes objetivos.

3 Trabalhos Relacionados

Prever o tempo de resolução de *issues* ou *bugs* é um assunto frequentemente abordado na literatura, em especial, devido à importância dessas estimativas para auxiliar o gerenciamento, alocação de recursos e definição dos prazos nos projetos. Existem estudos que propõem abordagens e modelos baseados no uso de aprendizado de máquina e técnicas de mineração de dados. Em geral, os modelos são específicos para algum sistema de *issue tracking* e utilizam técnicas adequadas às suas características. Boa parte das pesquisas se concentram no contexto *Open Source*, devido a maior facilidade para aquisição dos dados do que em projetos privados. Todavia, poucos estudos se referem ao desenvolvimento de modelos preditivos para o sistema de *issue tracking* do GitHub. Embora com um propósito diferente, outros trabalhos relevantes são aplicados ao sistema do GitHub e fornecem resultados interessantes, trazendo informações sobre o contexto do desenvolvimento na plataforma que podem ser úteis na construção de modelos preditivos.

Os trabalhos encontrados na literatura estão descritos em duas seções, uma no âmbito de modelos para estimar o tempo de resolução (Seção 3.1), e a outra no que compete a estudos aplicados sobre os dados presentes no GitHub (3.2).

3.1 Estimativa do tempo de resolução das issues

No contexto open source, a maioria dos trabalhos abordam o Bugzilla, por sua popularidade, como fonte dos dados para as estimativas. A pesquisa de Akbarinasaji, Caglayan e Bener (2018) se destaca por desenvolver uma réplica de um trabalho realizado por Zhang, Gong e Versteeg (2013) com dados de projetos privados e aplicá-la em um projeto *open source*. O intuito é investigar se a diferença de contexto impacta nos resultados. O foco é a criação de modelos para responder 3 tipos de perguntas: "Quantos *Bugs* podem ser resolvidos em uma determinada quantidade de tempo?", "Quanto tempo é necessário para resolver uma determinada quantidade de *bugs*?", "Quanto tempo é necessário para resolver um *bug* em particular?". No âmbito da última pergunta, o problema é tratado como classificação, onde o tempo de resolução é dividido nas classes "Rápido"e "Lento". São considerados somente os dados presentes no momento de criação dos *Bugs* e é utilizado o algoritmo *k-nearest neighbors (KNN)* como classificador. Usando-se a métrica F-meansure, os resultados variam entre 0.56 a 0.73, sendo menores do que os presentes no trabalho original de Zhang, Gong e Versteeg (2013), indicando que há indícios de uma maior dificuldade ao se realizar previsões no contexto *open source*.

O trabalho de Habayeb et al. (2017) também possui o Bugzilla como fonte de dados e faz proveito dos 6 diferentes status que podem ser atribuídos a um *bug*. Os

autores utilizam um *Hidden Markov Model* (HMM) para prever o tempo de resolução. HMMs são modelos baseados nos estados possíveis e transições entre eles, e as observações associadas aos estados e que ocorrem em uma ordem temporal. Os *bugs* são transformados em uma sequência de atividades ao longo do tempo e a ideia é que sequências similares demandam períodos de tempo similares. O tempo de resolução também é dividido nas classes "Rápido"e "Lento". Para extrair os conjuntos de teste são definidos 3 pontos de observação: 0, 3 e 7 dias desde a criação do *Bug*. Os resultados possuem a acurácia variando entre 0.60 a 0.75 e o F-score entre 0.59 a 0.77. Pombo e Teixeira (2020) realizam um trabalho similar, utilizando do mesmo modo um modelo HMM, sequências temporais e pontos para observar os dados. De forma similar aos trabalhos discutidos anteriormente, a abordagem desta pesquisa também considera eventos que ocorrem ao longo do tempo e define pontos para observar os dados. Contudo esses dados dinâmicos não são utilizados como sequências temporais.

Lee et al. (2020) também representam os *Bugs* como sequências de eventos ocorridos ao longo do tempo, entretanto, destacam-se por utilizar técnicas de aprendizagem profunda (*Deep learning*). Os autores ressaltam que não é trivial atingir alta acurácia no problema de prever o tempo de resolução de *issues*. Eles constroem um modelo com duas etapas: uma rede neural mista para extrair dados contextuais e uma rede de sequências para explorar as sequências temporais. O tempo de resolução é dividido em classes considerando várias alternativas para divisão, de 2 a 9 classes. A acurácia varia de 0.65 a 0.73 quando são consideradas apenas 2 classes, mas piora à medida que o número de classes aumenta, chegando a variar de 0.23 a 0.24 quando 9 classes são consideradas. Assim como em Lee et al. (2020), também são usadas redes neurais neste trabalho, por serem uma boa alternativa para problemas complexos. Contudo, não são utilizadas redes mistas uma vez que não são considerados dados dinâmicos na forma de sequências temporais.

Todos os trabalhos citados consideram o conteúdo textual dos *Bugs*, embora o foco sejam dados inerentes ao próprio Bugzilla, especialmente os estados dos *bugs*, que por si só já fornecem características importantes. Além de outros dados como prioridade do *Bug* [1], severidade [2], tipo [3], dentre outros. O trabalho de Ardimento, Boffoli e Mele (2020) se difere dos demais por desenvolver uma abordagem focada nos textos para prever o tempo de resolução, mas sem desconsiderar os outros dados. O texto é representado como *bag-of-words* e uma matriz TF-IDF é construída. Posteriormente, aplica-se sobre a matriz o algoritmo de redução de dimensionalidade *Principal Component Analysis* (PCA). Essa sequência de aplicação de técnicas também é conhecida como *latent semantic analysis* (LSA). O problema é modelado tanto como classificação quanto regressão, sendo utilizado o algoritmo *Support Vector Machine* (SVM) para prever as duas possíveis classes, e o

[1] *critical, serious, medium, minor*
[2] *blocking, functional, enhancement, cosmetic, orrequest*
[3] *defect, enhancement, task*

M5P model tree para prever os valores numéricos. A métrica RMSE é utilizada para avaliar o modelo de regressão e os resultados variam entre 123.33 a 152.02. O modelo de classificação é avaliado por meio da acurácia, com os resultados variando entre 0.74 a 0.78. Similar a Ardimento, Boffoli e Mele (2020), nesta pesquisa também existe um foco maior no conteudo textual. Todavia, será aplicada uma técnica alternativa ao LSA, a *latent dirichlet allocation* LDA.

Outro trabalho que trata o problema como regressão é o de Al-Zubaidi et al. (2017), mas usa o Jira como fonte de dados. O trabalho propõe uma abordagem baseada em busca e multi objetivo, onde são aplicados algoritmos evolucionários com o objetivo de aumentar a acurácia e diminuir a complexidade. O tempo de resolução é estimado em dias e a métrica *Standardized Accuracy* é utilizada para avaliar o modelo, com os resultados variando entre 53.8 a 63.2. O conjunto de dados utilizado não é grande, possuindo 8.260 *bugs* de 5 projetos diferentes.

Todos esses trabalhos diferem desta pesquisa em relação ao conjunto de dados, pois treinam e testam os modelos em cada projeto, sem considerar um conjunto com vários projetos. É provável que essa separação favoreça o desempenho dos modelos, uma vez que permite o aprendizado de características específicas de cada projeto. Todavia, dificulta a aplicação dos modelos em projetos novos e com poucos dados. Além disso, em geral o tamanho do conjunto de dados considerado nos trabalhos não é grande, sendo que o maior dentre eles possui 158831 *Bugs* de somente um projeto (AKBARINASAJI; CAGLAYAN; BENER, 2018).

Kikas, Dumas e Pfahl (2016), por sua vez, considera um conjunto de dados com mais de 900 mil *issues* e foi o único trabalho encontrado que desenvolve modelos de estimativa do tempo de resolução de *issues* para o GitHub. Os modelos utilizam duas categorias de atributos: dinâmicos, que mudam ao longo do tempo, e contextuais, que refletem o contexto do projeto. São definidos pontos para observar os dados, de forma similar aos outros trabalhos que lidam com a dinamicidade dos dados. O algoritmo *Random forest* é utilizado como classificador e prevê se a *issue* irá fechar ou não antes de um determinado prazo. O conjunto de treino considera vários projetos e os testes possuem o F-score variando de 0.23 a 0.89. A precisão ainda parece ser uma questão a ser trabalhada nos modelos dos autores, que apesar de informarem que ela é baixa, não fornecem os valores obtidos.

O trabalho de Kikas, Dumas e Pfahl (2016) é usado como base para esta pesquisa, por ser o único específico para o GitHub. Contudo conceitos presentes em outros trabalhos também são considerados, como o uso de redes neurais e a exploração de dados extraídos dos textos (ARDIMENTO; BOFFOLI; MELE, 2020; LEE et al., 2020). A fim de melhorar a qualidade das estimativas, novas categorias de atributos são propostas, os atributos temporais e textuais. Além disso, novos atributos são criados para as categorias

já existentes no trabalho de Kikas, Dumas e Pfahl (2016) e a fim de avaliar as melhorias, um modelo de referência é desenvolvido com base na descrição fornecida pelos autores e será descrito no Capítulo 6.

3.2 Trabalhos aplicados no GitHub

Apesar de não haver muitos estudos específicos sobre a estimativa do tempo de resolução de *issues* no GitHub, a plataforma é alvo de diversas pesquisas com outros contextos. Muitos trabalhos analisam os dados do GitHub a fim de identificar caraterísticas sobre o desenvolvimento úteis para outras pesquisas, como as que desenvolvem modelos preditivos para as ferramentas presentes na plataforma. Bissyandé et al. (2013) faz um estudo sobre o uso da plataforma e evidencia que projetos com mais *issues* tendem a ser mais famosos, possuindo mais *watchers*, *forks* e pessoas reportando *issues*. Esses projetos normalmente foram criados há mais tempo, possuem mais linhas de código e mais desenvolvedores tendem a colaborar. Todavia, apesar de serem os mais famosos, consistem em somente 8% do total, a maioria dos projetos não abrangem o desenvolvimento em equipe ou utilizam o GitHub só para hospedar código. Zhang et al. (2015) estuda o uso de menções e descobre que *issues* com mais menções tendem a ter mais comentários e desenvolvedores, entretanto, são mais difíceis e levam mais tempo para serem resolvidos. Destefanis et al. (2018) constroem uma rede de colaboração com base nas *issues* e seus comentários. Os colaboradores são divididos em desenvolvedores e comentaristas, aqueles que nunca fizeram *commits*, e os resultados confirmam que eles consistem em grupos distintos que se comportam de maneira diferente.

No âmbito de estudos com base em aprendizado de máquina, existem abordagens desenvolvidas para diversos fins. Zhang et al. (2020) cria modelos para conectar *issues* potencialmente relacionadas, com o propósito de melhorar a aquisição de conhecimento sobre as questões e fornecer informações relacionadas ao pesquisar e resolvê-las. Sorbo et al. (2019) desenvolvem uma abordagem para rotular automaticamente *issues* que não serão resolvidas, com a motivação que desenvolvedores demoram em média 5 meses para identificar que não será possível resolver uma questão. Além disso, investigam as características desse tipo de *issues*. Dhasade, Venigalla e Chimalakonda (2020) realizam um estudo preliminar para priorizar *Pull requests*, incluindo uma pesquisa com desenvolvedores para descobrir os principais fatores que devem ser levados em consideração, e com base neles um modelo é desenvolvido para realizar a priorização de forma automatizada. Um dos fatores apontados na pesquisa é o tempo demandado para resolver um problema, sendo ressaltado pelos autores que estimar o tempo de resolução é útil para o processo de priorização.

Os trabalhos de análise de dados do GitHub são importantes para esta pesquisa no âmbito da caracterização do contexto onde o problema está inserido. As informações evi-

denciadas por Zhang et al. (2015) e Destefanis et al. (2018) servirão para propor atributos e extrair informações relacionadas às *issues*. Da mesma forma que os estudos anteriores, este trabalho também desenvolve modelos de aprendizado de máquina. Uma vez que todos os trabalhos citados possuem o GitHub como base, as contribuições desenvolvidas neste trabalho também são úteis para essas outras pesquisas. Por exemplo, fornecendo informações para aprimorar modelos similares aos dos trabalhos de Zhang et al. (2020) e Sorbo et al. (2019) tanto com as estimativas do tempo de resolução quanto com atributos propostos.

4 Metodologia

O desenvolvimento de projetos na área ciência dos dados é comumente dividido em etapas, de acordo com os objetivos e o problema a ser resolvido. A utilização de uma metodologia adequada define um caminho a ser seguido e torna o processo de desenvolvimento mais organizado. Este capítulo descreve a metodologia aplicada durante o desenvolvimento dos modelos preditivos presentes neste trabalho.

Metodologias como a Cross Industry Standard Process for Data Mining (CRISP-DM) e Team Data Science Process (TDSP)[1] formalizam o processo para obtenção de soluções envolvendo análise preditiva Wirth e Hipp (2000), Microsoft (2020). A TDSP, lançada em 2016 pela Microsoft, consiste em um refinamento das melhores práticas da Microsoft e outras indústrias a fim de auxiliar a implementação bem-sucedida de iniciativas de ciência de dados. O projeto desenvolvido neste trabalho utiliza uma metodologia baseada no ciclo de vida da TDSP, porém de forma simplificada devido ao tamanho da equipe e por se tratar de um projeto acadêmico.

4.1 A Adaptação da TDSP

A TDSP é uma metodologia ágil e iterativa, focada no desenvolvimento de projetos de ciência dos dados sendo parte de um produto, como softwares que utilizam algum componente com inteligência artificial. Todavia, de acordo com a Microsoft, a metodologia também pode ser utilizada para ciência de dados exploratória, mas sem a necessidade de algumas etapas.

A Figura 1 apresenta o diagrama organizacional do ciclo de vida adotado neste trabalho. A TDSP estrutura o ciclo em quatro etapas: entendimento do negócio, aquisição e compreensão dos dados, modelagem e implantação. A última etapa consiste na implantação do modelo desenvolvido em uma linha de produção. Entretanto, a implantação não é relevante para esta pesquisa, uma vez que não se trata de uma execução na industria, mas sim no meio acadêmico. Sendo assim, neste trabalho a etapa de disponibilização substitui a implantação, pois uma vez disponibilizado o material desenvolvido a implantação pode ser executada futuramente caso algum membro da comunidade tenha interesse.

Uma das vantagens da organização do ciclo de vida proposto pela TDSP é a possibilidade de alternar livremente entre as etapas, como é indicado pelas setas na Figura 1. Por exemplo, na etapa de modelagem pode ser identificada a necessidade de melhorar a compreensão do negócio ou dos dados, e essas etapas podem ser retomadas. Cada etapa

[1] https://docs.microsoft.com/pt-br/azure/machine-learning/team-data-science-process/lifecycle

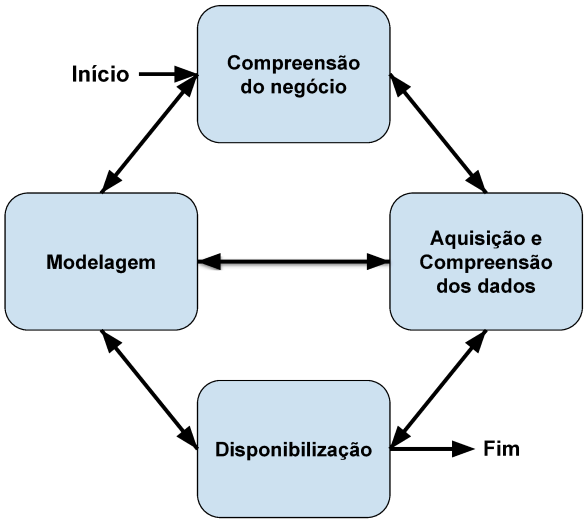

Figura 1 – Metodologia adotada durante o projeto, baseada na TDSP (MICROSOFT, 2020)

possui objetivos e tarefas específicas e os detalhes sobre elas são apresentados nas próximas subseções.

4.1.1 Compreensão do negócio

A etapa de compreensão do negócio é focada no entendimento do problema e seus objetivos, bem como o contexto onde ele está inserido. As metas dessa etapa consistem em: especificar as variáveis e métricas relacionadas para determinar o sucesso do projeto, e identificar as fontes dos dados relevantes. Para atingir essas metas, as seguintes tarefas devem ser executadas:

- **Definir os objetivos**: identificar e compreender o problema, formular perguntas a serem respondidas, com base na definição de quais variáveis precisam ser previstas. Identificar métricas relacionadas às variáveis que permitam determinar o sucesso do projeto, por exemplo, atingir a precisão de "X" por cento.
- **Identificar as fontes de dados**: Encontrar as fontes dos dados relevantes, que ajudem a responder as perguntas e atingir as metas, forneçam informações relacionadas às variáveis e exemplos de respostas para as perguntas.

Neste trabalho, o objetivo é estimar o tempo de resolução de *issues*, sendo esse tempo a variável a ser prevista, e as perguntas a serem respondidas são definidas nas hipóteses da pesquisa (Capítulo 1). O contexto onde o problema está inserido é descrito no referencial teórico (Capítulo 2). A fonte dos dados é o GitHub e a meta é superar

o modelo considerado estado da arte para previsão do tempo de resolução de *issues* no GitHub Kikas, Dumas e Pfahl (2016).

4.1.2 Aquisição e compreensão dos dados

O objetivo desta etapa é produzir um conjunto de dados suficiente para resolver o problema. A fim de alcançar esse objetivo, a seguinte meta é definida: Produzir um conjunto de dados limpo e de alta qualidade, onde a relação dos dados com a variável a ser prevista seja compreendida. Sendo assim, as seguintes tarefas devem ser executadas:

- **Aquisição dos dados**: Configurar o processo para mover os dados da fonte de origem para o local onde serão executadas as operações de análise.
- **Exploração dos dados**: Auditar a qualidade dos dados, identificando e removendo discrepâncias como valores ausentes ou ruído. Essa tarefa também envolve compreender os padrões inerentes aos dados, como eles estão relacionados e se são suficientes e relevantes para resolução do problema.

A tarefa de aquisição dos dados consiste na implementação de consultas e *scripts* para extração dos dados necessários. Uma vez extraídos, a exploração verifica a qualidade e se os dados são adequados. Além disso, desenvolve-se uma melhor compreensão dos dados, sendo possível criar resumos e visualizações sobre eles. Ambas as tarefas costumam ser iterativas e incrementais, podendo, eventualmente, haver a necessidade de encontrar mais fontes e aumentar o conjunto de dados obtido inicialmente. O Capítulo 5 fornece detalhes sobre a execução dessa etapa e suas tarefas.

4.1.3 Modelagem

Com o conjunto de dados pronto, nesta etapa é desenvolvido o modelo de aprendizado de máquina. Para isso, devem ser atingidas as seguintes metas: determinar os atributos dos dados ideais para o modelo, e com base neles, criar um modelo que faça boas previsões. Dessa forma, as seguintes tarefas devem ser executadas:

- **Engenharia de atributos**: Criar atributos sobre os dados brutos para facilitar o treinamento do modelo. A engenharia de atributos envolve a agregação e a transformação das variáveis brutas de forma criativa, com base no domínio do problema e nas informações obtidas na etapa de exploração dos dados.
- **Treinamento do modelo**: Encontrar o modelo que responda à pergunta com mais precisão, comparando diferentes métodos e escolhendo o mais adequado. Para esse fim, deve-se dividir o conjunto de dados em partes para treino e teste, criar modelos

usando diferentes métodos, treiná-los com o conjunto de treino e avaliá-los com conjunto de teste.

A engenharia de atributos faz proveito das informações obtidas na etapa de "Aquisição e compreensão dos dados", a fim construir novos atributos que estejam relacionados com a variável a ser prevista. Uma vez definidos os atributos, os modelos são treinados e avaliados para se identificar o melhor. Nessa tarefa de treinamento do modelo, podem ser testados modelos construídos a partir de diferentes algoritmos, como *random forests* e redes neurais, sendo também possível avaliar as melhores configurações de cada algoritmo. A avaliação dos modelos ocorre com base nos objetivos definidos na etapa de "Compreensão do negócio"e utiliza métricas como acurácia e precisão e as demais descritas no referencial teórico (Seção 2.4). A avaliação também investiga se é necessário mais iterações das etapas anteriores, como experimentar abordagens alternativas ou adicionar mais dados e atributos.

Neste trabalho, escolhe-se um modelo de referência baseado no trabalho de Kikas, Dumas e Pfahl (2016) como fase inicial da etapa de modelagem. Com base na metodologia são avaliados os aprimoramentos com o intuito de aumentar a acurácia e precisão do modelo. As contribuições são desenvolvidas através de iterações das tarefas de engenharia de atributos e treinamento do modelo e são descritas no Capítulo 6.

4.1.4 Disponibilização

Como implantar o modelo não faz parte dos objetivos deste trabalho, a etapa de Disponibilização foca em tornar acessível o material desenvolvido. Sendo assim, a meta é: disponibilizar o código em um local público como repositórios de software Open Source, de forma compreensível e capaz de ser utilizado em pesquisas futuras. Por essa razão, as tarefas que devem ser executadas são:

- **Criação de documentações**: Criar documentos descrevendo os modelos e artefatos relacionados, com o intuito de facilitar a compreensão do material desenvolvido.
- **Disponibilização do material**: Hospedar o código fonte dos modelos e arquivos relacionados em um local público. Dentre os arquivos estão, *scripts* para aquisição dos dados e criação dos atributos, o conjunto de dados final e resultados de testes.

Como o trabalho utiliza dados sobre projetos no GitHub, a plataforma é um ótimo local para disponibilizar os materiais. Um repositório deste trabalho foi criado no GitHub com base na estrutura padrão de projetos da TDSP[2] e pode ser acessado no repositó-

[2] https://docs.microsoft.com/pt-br/azure/machine-learning/team-data-science-process/overview#standardized-project-structure

rio do GitHub de um dos autores[3]. Esta dissertação como um todo também faz parte da documentação, além de artigos descrevendo o modelo e documentações presentes no repositório.

[3] https://github.com/Luizvx/issue-lifetime-estimator

5 Conjunto de dados

Os dados são primordiais para uma pesquisa de mineração de dados. Portanto, o objetivo deste capítulo é descrever os processos realizados para obtenção do conjunto de dados necessário para o desenvolvimento do trabalho. A fonte e os passos para aquisição dos dados serão apresentados na Seção 5.1. A Seção 5.2 apresentará informações sobre a limpeza e filtros aplicados a fim de aumentar a qualidade do conjunto de dados.

5.1 Aquisição dos dados

Tendo o GitHub como fonte dos dados, é necessário definir como extrair os dados, e trazê-los para o ambiente de trabalho. O próprio GitHub fornece ferramentas para extração por meio de sua API Rest (GITHUB, 2020), onde todos os dados públicos podem ser adquiridos através de requisições Http. Entretanto, somente uma informação específica pode ser recuperada por requisição, como, por exemplo, informações de um único projeto, *issue* ou comentário. Como um projeto pode ter muitas *issues* e uma *issue* muitos comentários, muitas requisições são necessárias e aumenta-se a demanda de tempo para recuperar um projeto por completo utilizando a API. Tendo em vista essa questão, projetos como o GHtorrent foram criados, possibilitando a aquisição de grandes volumes de dados sobre o GitHub (GOUSIOS, 2013).

O GHtorrent (GOUSIOS, 2013), é utilizado neste trabalho como fonte dos dados a serem minerados. A construção da base de dados pelo GHtorrent ocorre por meio da utilização de *streams* de eventos públicos do GitHub, como a criação de uma *issue*, e da API Rest para extração e transferência das informações para um banco de dados. Os dados presentes vão até o dia 01 de junho de 2019, data em que o projeto foi descontinuado e a base parou de ser atualizada. O GHtorrent oferece várias formas de aquisição dos dados, como dumps do MySQL e MongoDB ou opção de consultar uma versão online da base hospedada no Google BigQuery. Este trabalho utiliza a versão da base no Google BigQuery, que contém informações sobre os projetos, commits, *issues*, *pull requests*, entre outras. Contudo, o conteúdo textual das *issues* e cometários não estavam presentes ao realizar o trabalho.

Considerando-se a relevância do conteúdo textual das *issues* e comentários para fins das análises desta pesquisa, fez-se necessário buscar alternativas de recuperação de tais informações. Inicialmente, foi reportada uma *issue* relatando o problema no repositório do projeto GHtorrent, na expectativa de que ela fosse atendida em tempo hábil. No entanto, como isso não aconteceu, foi implementado um extrator dos dados desejados, que funciona da seguinte forma:

1. Seleciona um projeto aleatório presente no Ghtorrent;
2. Faz a requisição na API do GitHub para cada *issue* do projeto;
3. Faz a requisição na API do GitHub para cada comentário da *issue*;
4. Salva o texto da *issue* e seus comentários em um banco de dados local.

A API utiliza uma conta do GitHub e possui um limite de 5000 requisições por hora, quando tal limite é atingido, a conta é trocada para não interromper a extração. O extrator foi executado até que o volume de dados atingisse pelo menos 900 mil *issues* e considera somente os projetos selecionados segundo os filtros apresentados na próxima seção.

5.2 Limpeza dos dados

A etapa de limpeza dos dados melhora a qualidade, evitando inconsistências, e selecionando somente as informações adequadas ao contexto do trabalho. Primeiramente, é definida uma data limite para os dados em 01 de janeiro de 2019, devido à parte mais recente (de janeiro a junho), estar incompleta. Essa falta de integridade ocorre por existir um atraso na atualização da base pelo Ghtorrent e nem todas as informações estarem instantaneamente visíveis. Posteriormente, foram implementados os mesmos filtros propostos por Kikas, Dumas e Pfahl (2016), mas com algumas particularidades referente as *issues*. O objetivo é tornar os modelos desenvolvidos neste trabalho mais comparáveis aos deles. Os detalhes sobre cada filtro aplicado serão mencionados nas próximas subseções.

5.2.1 Filtros dos projetos

Existem em torno de 125 milhões de projetos registrados na versão mais recente do GHtorrent. Entretanto muitos deles utilizam o GitHub somente para hospedagem de código ou não são projetos de software, por exemplo, projetos que hospedam somente documentações (KALLIAMVAKOU et al., 2014). Como a pesquisa é voltada para projetos de desenvolvimento de software em equipe, os filtros possuem o propósito de evitar projetos muito pequenos, sem equipe ou com pouca atividade de *issue tracking*. Sendo assim, os projetos são selecionados a partir dos seguintes filtros:

- **FP1**: Ter a data de criação no intervalo de 1 de janeiro de 2012 a 30 de Abril de 2018;
- **FP2**: Possuir pelo menos 8 meses de histórico de issues;
- **FP3**: Ter pelo menos 100 *issues* abertas e ao menos 1 fechada;

- **FP4**: Possuir 5 *commits* ou mais;
- **FP5**: Não ter criado mais de 2000 issues por mês ou mais de 500 *issues* por dia;
- **FP6**: Não ser *fork* de outro projeto;
- **FP7**: Não possuir *commits* antes da criação.

A importância do filtro FP1 advém do fato que projetos criados antes 2012 possuem os dados parcialmente disponíveis no GHtorrent. Por outro lado, projetos após 30 de Abril de 2018 não possuem um histórico de 8 meses antes da data limite. Esse histórico de 8 meses também é garantido pelo filtro FP2, a fim de evitar projetos com pouco tempo de atividade. O filtro FP3 seleciona projetos que utilizam ativamente o sistema de *issue tracking*. Projetos com pouca atividade de versionamento de código são excluídos pelo FP4. Alguns projetos usam automação para a criação das *issues*, sendo estes excluídos pelo FP5. Os filtros FP6 e FP7 excluem projetos que são *fork*, ou seja, cópias de outro projeto.

5.2.2 Filtros das issues

Com o objetivo de evitar inconsistências e não usar dados fora do contexto da pesquisa, foram selecionadas somente as *issues* que atendem aos seguintes filtros:

- **FI1**: Ter a data de criação no intervalo de 1 de janeiro de 2012 a 31 de dezembro de 2018;
- **FI2**: Não possuir eventos ou comentários antes da data de criação;
- **FI3**: Não possuir evento de reabertura;
- **FI4**: Não ser *pull request*;

O filtro FI1 seleciona somente *issues* criadas desde a data em que os dados se encontram completos até a data limite. Algumas *issues* possuem eventos ou comentários criados antes ou ao mesmo tempo que a *issue*, sendo estas removidas pelo filtro FI2. Embora Kikas, Dumas e Pfahl (2016) não removam tais *issues*, este filtro foi considerado porque não faz sentido criar eventos ou comentários para uma *issue* que ainda não foi reportada e, portanto, ainda não existe. *Issues* podem ser reabertas e fechadas novamente, contudo o escopo desta pesquisa se limita a *issues* que não foram reabertas, e por esta razão o filtro FI3 é aplicado.

Ao investigar a documentação do Ghtorrent é possível perceber que as *pull requests* são tratadas como uma extensão das *issues*. Isso ocorre pois *pull requests* possuem as mesmas informações, mas com a adição de um trecho de código. Sendo assim, elas

aparecem na tabela de *issues* com um flag indicando que é uma *pull request*, e por essa razão o filtro FI4 é aplicado. Kikas, Dumas e Pfahl (2016) não mencionam se removem ou não as *pull requests*, todavia, a estimativa de tempo delas é uma tarefa distinta e tratada em outras pesquisas na literatura (MADDILA; BANSAL; NAGAPPAN, 2019; YU et al., 2015). Diferentemente das *issues*, *pull requests* já apresentam uma possível solução por meio do código e só são resolvidas quando o código atrelado é revisado e aprovado para ser mesclado ao repositório.

5.3 Considerações finais

As etapas de aquisição dos dados garantem a completude do conjunto de dados, obtendo todas as informações necessárias. O processo de limpeza torna o conjunto específico para o contexto da pesquisa e reduz inconsistências nos dados. Ao todo, os filtros selecionam 21.495 projetos, mas faz parte do conjunto somente a amostra aleatória de projetos recuperados pelo extrator. Após a limpeza, o conjunto final possui 2.913 projetos, totalizando 917.382 *issues*, das quais 671.353 (73%) foram fechadas e 246.029 (27%) não foram fechadas antes da data limite. Tendo em vista os fatos mencionados, o conjunto completo e limpo possui uma quantidade suficiente de *issues* e torna possível o desenvolvimento de um modelo de aprendizado de máquina.

6 Desenvolvimento dos Modelos de Previsão

Uma vez que o conjunto de dados encontra-se no ambiente de trabalho e o pré-processamento está concluído, torna-se viável o desenvolvimento de modelos de aprendizado de máquina. Como o problema abordado é um problema de previsão, os modelos são responsáveis por realizar de forma automatizada as estimativas do tempo de resolução das *issues*. Para tal, é necessário modelar o problema e aplicar os algoritmos de aprendizado de máquina. Os algoritmos aprendem padrões sobre dados do passado e realizam previsões dos dados futuros aplicando esses padrões. A fim de se obter boas estimativas, são criados atributos sobre os dados brutos com o intuito de fornecer mais informações relevantes sobre as *issues*. Além disso, os algoritmos devem ser configurados e comparados com o propósito de se escolher o mais adequado.

A Seção 6.1, apresenta o trabalho escolhido como referência para construção dos modelos e descreve a forma como o problema é modelado. Grande parte dos aprimoramentos realizados estão concentrados na criação de novos atributos e o processo para concepção deles é descrito na Seção 6.2. Outra melhoria é a escolha e configuração de algoritmos mais robustos para realizar as estimativas. A Seção 6.3 discute essa escolha e descreve os passos para o treinamento dos modelos, como a divisão dos conjuntos para treino, teste e validação.

6.1 Desenvolvimento do modelo referência

A primeira fase da etapa de Modelagem (Capítulo 4) consiste no desenvolvimento de um modelo de referência, que pode ser adaptado, refinado e ajustado para que se possa melhorar a qualidade das estimativas. Os modelos desenvolvidos por Kikas, Dumas e Pfahl (2016) são escolhidos como referência para esta pesquisa. Deste modo, foram criados modelos de acordo com a descrição fornecida pelos autores, sendo mantida a forma como o problema foi modelado por eles e explicada nesta Seção.

Tendo em vista a carência de dados sobre as *issues* no sistema de *issue tracking* do GitHub, onde *issues* normalmente só possuem informações textuais, Kikas, Dumas e Pfahl (2016) faz o uso de atributos dinâmicos e contextuais em seus modelos. Os atributos contextuais fornecem informações sobre o estado do projeto, fazendo proveito da integração com o sistema de controle de versão do GitHub. Os atributos dinâmicos referem-se a características que mudam ao longo do tempo, como a quantidade de comentários em uma *issue*, por exemplo. Para viabilizar a utilização de atributos dinâmicos, a modelagem envolve alguns conceitos importantes como a definição de pontos de observação e horizontes de tempo.

6.1.1 Ponto de observação e Horizonte de tempo

Como os atributos dinâmicos mudam com o passar do tempo, deve ser imposto um momento para os dados serem observados. O momento é definido pelo **ponto de observação** (PO), que consiste na quantidade de dias decorridos desde a criação da *issue*. Por exemplo, considerando o ponto de observação igual a 7 dias, o atributo quantidade de comentários irá considerar somente os comentários criados a partir do momento em que a *issue* foi criada até o sétimo dia. Todos os atributos dinâmicos seguem a mesma lógica, tendo o ponto de observação como data limite para os dados. Tal definição simula o contexto real em que o modelo será aplicado. Ao selecionar um projeto em algum momento, serão encontradas *issues* abertas com diversos tempos de vida, algumas podem ter sido reportadas recentemente (PO igual a 0), outras se encontram abertas há um tempo (PO maior que 0).

Estimar o tempo de resolução pode ser tratado como um problema de classificação ou regressão. Contudo, em problemas de regressão é mais difícil realizar estimativas precisas. Uma simplificação viável é a divisão do tempo de resolução em classes, tratando o problema como classificação. A classificação se torna binária com a utilização de um **horizonte de tempo** (HT), que consiste no prazo em que se deseja saber se uma *issue* será resolvida ou não. Por exemplo, considerando o HT igual a 30 dias, todos as *issues* que foram resolvidas antes de 30 dias pertencerão à classe positiva (classe 1), e as que não foram resolvidas à classe negativa (classe 0).

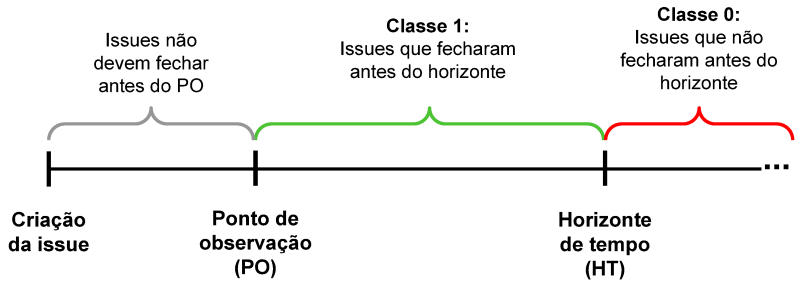

Figura 2 – Ponto de observação e horizonte de tempo representados na linha do tempo das issues

A Figura 2 ilustra os conceitos Ponto de observação e Horizonte de tempo, por meio da linha do tempo de uma *issue*. Ao utilizar esses conceitos procura-se responder perguntas do tipo "Uma *issue* criada a PO dias será resolvida em HT dias?". Se sim a *issue* será fechada antes do horizonte e deverá ser prevista como pertencente à classe 1, ou à classe 0 caso o contrário. Uma vez definido um ponto de observação, uma *issue* deve durar até ele, ou seja, não pode ter sido fechada antes, caso o contrário não haveria dados suficientes.

6.1.2 Criação dos modelos

Kikas, Dumas e Pfahl (2015) criam modelos variando combinações de pontos de observação e horizontes de tempo, sendo escolhidos de acordo com períodos do calendário: dia, semana, quinzena, mês, trimestre, semestre e ano. Dessa forma, são definidos 7 pontos de observação (0, 1, 7, 14, 30, 90 e 180 dias) e 7 horizontes de tempo (1, 7, 14, 30, 90, 180, e 365 dias), sendo o ponto de observação 0 correspondente ao momento em que a *issue* foi criada. Um modelo é criado para cada combinação de PO e HT para prever se a *issue* irá fechar ou não antes do horizonte. As combinações só fazem sentido se o horizonte for maior que o ponto de observação, dessa forma, somente 28 combinações são válidas e essa é quantidade de modelos criados. Por exemplo para o PO 30, os HTs válidos são somente 90, 180 e 365.

Em modelos com ponto de observação zero, alguns atributos dinâmicos não são significativos, pois consideram-se somente dados até o momento da criação das *issues*. Modelos com o ponto de observação maior que zero, adicionam uma restrição nos dados (*issues* não podem fechar antes do PO), e, consequentemente, o tamanho do conjunto de dados varia. Quanto maior o ponto de observação, mais restritos os dados são e menor é o conjunto. Por exemplo, modelos com o PO igual a 30 consideram somente *issues* que duraram 30 dias. Essa forma de modelagem criando vários modelos permite uma melhor investigação da importância de cada atributo e da precisão de cada caso. Previsões de curto prazo, com o horizonte de 1 ou 7 dias por exemplo, podem ser mais difíceis.

Os atributos criados por Kikas, Dumas e Pfahl (2016) foram replicados com base na descrição fornecida, com exceção do *text score*. O *text score* mede uma pontuação para o texto presente no título e na descrição das *issues*, mas a forma como o atributo é calculado não é bem descrita. Apesar desse atributo não ter sido implementado, vários outros atributos derivados do texto são propostos neste trabalho. Além dos atributos textuais, também são propostos outros atributos contextuais, dinâmicos e temporais. Cada uma dessas categorias e seus respectivos atributos são explicados na próxima seção.

6.2 Engenharia de atributos

Os atributos fornecem características sobre os dados para os modelos. O processo de engenharia de atributos transforma dados brutos em informações relevantes, com o objetivo de fornecer características melhores e consequentemente melhorar as estimativas realizadas. Esse processo envolve o conhecimento do domínio onde o problema está inserido, sendo que tal conhecimento pode ser adquirido na exploração dos dados. Com base nessa experiência os atributos simples são transformados e agregados em equações que calculam atributos compostos e robustos.

Os dados brutos sobre as *issues* em si não são muitos e em geral se resumem ao

título, descrição, autor e data. Portanto, o processo de engenharia de atributos é bastante importante nesse contexto, uma vez que existem dados relacionados às *issues*, tais como autor, equipe e projeto, que podem se tornar informações úteis para os modelos. Tendo em vista os fatos mencionados, os atributos propostos são organizados nas categorias descritas nas próximas subseções.

6.2.1 Atributos dinâmicos

A partir do momento em que uma *issue* é criada, várias mudanças podem acontecer, tanto na *issue* em si quanto no projeto. Com o passar do tempo, a *issue* pode receber mais atenção, sendo postados comentários em sua discussão, ou sendo citada em um *commit*, entre outros. O projeto, por sua vez, pode passar por uma fase de alta ou baixa produtividade dos colaboradores e aumentar ou diminuir o tamanho da equipe. Todas essas variáveis mudam no decorrer do tempo e são representadas pelos atributos dinâmicos. Essas mudanças condizem com a dinamicidade dos grupos de desenvolvedores (Core e Co-desenvolvedores), e justificam a relevância desses atributos.

A aplicação dos atributos dinâmicos ocorre por meio do ponto de observação. Ao calcular esses atributos devem ser considerados somente os eventos ocorridos dentro de um intervalo de tempo, cujo limite é o ponto de observação. O início desse intervalo, em geral, é a data da criação da *issue*, embora esse início possa variar. Como em todas as outras categorias existem atributos dinâmicos, os que não se encaixam em outras categorias são apresentados na Tabela 1 e os outros aparecerão nas próximas tabelas identificados com um "T"no final do nome.

Tabela 1 – Atributos dinâmicos que não se encaixam em outras categorias

Atributo	Descrição
Eventos da issue	
nAssignmentsT	Kikas, Dumas e Pfahl (2016): Quantidade de eventos de atribuição desde a criação da issue até o PO.
nLabelsT	Kikas, Dumas e Pfahl (2016): Quantidade de labels adicionadas desde a criação da issue até o PO.
nMentionedByT	Kikas, Dumas e Pfahl (2016): Número de vezes que a issue foi mencionada em outras issues, no período da criação até o PO.
nReferencedByT	Kikas, Dumas e Pfahl (2016): Número de vezes que a issue foi mencionada em mensagens de commit, no período da criação até o PO.
nSubscribedByT	Kikas, Dumas e Pfahl (2016): Quantidade de pessoas que subscreveram na issue para receber atualizações desde a criação até o PO.
nCommentsT	Kikas, Dumas e Pfahl (2016): Quantidade de comentários postados desde a criação da issue até o PO.
Participantes da issue	
nActorsT	Kikas, Dumas e Pfahl (2016): Quantidade de atores, pessoas que comentaram, referenciaram ou subscreveram a issue, no período de 2 semanas antes da criação até o PO.
nParticipantsT	Quantidade de pessoas que postaram algum comentário na issue, no período de 2 semanas antes da criação até o PO.

6.2.2 Atributos contextuais

No momento da criação de uma *issue*, vários eventos podem estar ocorrendo no projeto e consequentemente afetarem seu tempo de resolução. Esse contexto momentâneo sobre o projeto onde a *issue* se encontra é representado pelos atributos contextuais. Esses atributos referem-se ao estado do projeto, como a quantidade de código que está sendo produzida ou a quantidade de *issues* sendo reportadas ou resolvidas. Por exemplo, uma *issue* reportada simultaneamente a outras, pode demorar mais tempo para ser resolvida, pois pode haver uma sobrecarga da equipe de desenvolvedores. Essa informação em específico se refere ao contexto no próprio sistema de *issue tracking*. Contudo, alguns dos atributos contextuais se aproveitam da integração com as ferramentas de versionamento de código e fornecem informações sobre o andamento do desenvolvimento.

A Tabela 2 apresenta os atributos contextuais com informações sobre as *issues*, seus autores e participantes. Os atributos sobre o autor metrificam a participação dele no projeto, fazendo proveito da características do core (desenvolvedores principais). Se o autor fizer parte do core provavelmente fará muitos *commits* no projeto, sendo essa informação capturada pelo atributo ”nCommitsByCreator”. Membros do core costumam ter um envolvimento maior na comunicação e o atributo ”reporterParticipatedIssuesRate” reflete essa característica. Os membros do core também possuem um grande conhecimento sobre o projeto, por essa razão é provável que suas *issues* sejam relevantes podendo ser resolvidas mais rapidamente, tal fato justifica o atributo ”reporterIssueTimeMean” (CROWSTON et al., 2006). O atributo ”memberReporter” se refere aos membros da organização, um grupo que não é equivalente ao core e muitas vezes se restringe aos proprietários do projeto. Ao todo, 3 atributos referentes aos autores foram aproveitados do trabalho de Kikas, Dumas e Pfahl (2016), 1 do trabalho de Hooimeijer e Weimer (2007), e 4 foram propostos nesta pesquisa.

O conceito por trás dos atributos referentes aos participantes da *issue* é similar, ou seja, também fazem proveito das características do core. Alguns deles metrificam quantos participantes estão envolvidos com o desenvolvimento e utilizam os sistemas (”nCommitsByUniqueActorsT”, ”nParticipantsDoPullsT”, ”nParticipantsReportIssuesT”). Outros medem o quanto esses participantes estão envolvidos (”nParticipantPullsT”, ”nParticipantIssuesT”). No total, 4 atributos foram reaproveitados do trabalho de Kikas, Dumas e Pfahl (2016) e 6 foram propostos, como pode ser visto na tabela 2.

Tendo em vista a dinamicidade do core de desenvolvedores, todos os atributos consideram somente os dados em um intervalo de tempo específico. A importância desse intervalo é explicada pois um desenvolvedor que já foi bastante ativo em algum momento pode diminuir a quantidade de contribuições, ou vice-versa (CROWSTON et al., 2006). Dessa forma, é garantido que os atributos reflitam o contexto mais recente (atual).

A Tabela 3 apresenta os atributos contextuais com informações sobre o projeto.

Tabela 2 – Atributos contextuais derivados de dados sobre as issues, seus autores e participantes

Atributo	Descrição
Issue	
Assigned	Se a issue foi atribuída a alguém
Autor da issue	
nIssuesByCreator	Kikas, Dumas e Pfahl (2016): Quantidade de issues reportadas pelo autor nos 3 meses antes da criação da issue.
nIssuesByCreatorClosed	Kikas, Dumas e Pfahl (2016): Quantidade de issues reportadas pelo autor que foram resolvidas nos 3 meses antes da criação da issue.
nCommitsByCreator	Kikas, Dumas e Pfahl (2016): Quantidade de commits realizados pelo autor nos 3 meses antes da criação da issue.
HooimeijersReporterReputation	Hooimeijer e Weimer (2007): Proporção de issues criadas pelo autor que foram resolvidas, considerando o período de 3 meses antes da criação da issue.
reporterIssueTimeMean	Tempo médio de resolução das issues reportadas pelo autor no período de 3 meses antes da criação da issue.
reporterParticipatedIssuesRate	Proporção de issues no projeto que o autor postou comentários, considerando o período de 3 meses antes da criação.
memberReporter	Se o autor é membro da organização do projeto.
reporterCommitsRate	Proporção de issues no projeto que o autor postou comentários, considerando o período de 3 meses antes da criação.
Participantes da issue	
nCommitsByActorsT	Kikas et al. (2016): Quantidade de commits realizados pelos atores, no período de 2 semanas antes da criação até o PO.
nCommitsByUniqueActorsT	Kikas et al. (2016): Quantidade de atores que fizeram commit no repositório no período de 2 semanas antes da criação até o PO.
nParticipantsDoPullsT	Quantidade de participantes que criaram pull requests, no período de 2 semanas antes da criação até o PO.
nParticipantsReportIssuesT	Quantidade de participantes que reportaram issues, no período de 2 semanas antes da criação até o PO.
nMemberParticipantsT	Quantidade de participantes de que são membros da organização, no período de 2 semanas antes da criação até o PO.
nParticipantPullsT	Quantidade de pull requests criadas pelos participantes, no período de 2 semanas antes da criação até o PO.
nParticipantIssuesT	Quantidade de issues criadas pelos participantes, no período de 2 semanas antes da criação até o PO.

Esses atributos também consideram um período de tempo específico e possuem o propósito de representar a quantidade de atividade no projeto. Projetos podem passar por picos de atividade, com muitas pessoas envolvidas e uma quantidade expressiva de código sendo desenvolvido. Uma *issue* criada nesse momento pode ser resolvida mais rápido do que se fosse criada em um período de inatividade. Além disso, não existem informações explícitas que um projeto está inativo ou foi cancelado e *issues* criadas em projetos inativos tendem a não ser resolvidas. Entretanto, em projetos inativos não haverá *commits* recentes e essa informação é obtida pelos atributos. Ao todo, são usados 13 atributos contextuais sobre o projeto, sendo 6 propostos por (KIKAS; DUMAS; PFAHL, 2016) e 7 por este trabalho.

Tabela 3 – Tabela de atributos contextuais com informações sobre o projeto

Atributo	Descrição
Projeto	
nIssuesCreatedInProject	Kikas et al.: Quantidade de issues criadas no projeto, nos 3 meses antes da criação da issue.
nIssuesCreatedInProjectClosed	Kikas et al.: Quantidade de issues criadas e fechadas no período de 3 meses antes da criação da issue.
nCommitsInProject	Kikas et at.: Quantidade de commits realizados no projeto, nos 3 meses antes da criação da issue.
nIssuesCreatedProjectT	Kikas et al.: Quantidade de issues criadas no projeto, no período de 2 semanas antes da criação até o PO.
nIssuesCreatedProjectClosedT	Kikas et al.: Quantidade de issues fechadas no projeto, no período de 2 semanas antes da criação até o PO.
nCommitsProjectT	Kikas et al.: Quantidade de commits realizados no projeto, no período de 2 semanas antes da criação até o PO.
projectIssueTimeMean	Média do tempo de resolução das issues fechadas no projeto, nos 3 meses antes da criação.
nTotalOpenIssuesInProjectT	Total de issues abertas no projeto, desde o início até a data de criação da issue.
nTotalClosedIssuesInProjectT	Total de issues fechadas no projeto, desde o início até a data de criação da issue.
nClosedIssuesLifetimeT	Quantidade de issues fechadas durante o tempo de vida da issue, ou seja, desde a criação até o PO.
nOpenIssuesLifetimeT	Quantidade de issues abertas durante o tempo de vida da issue, ou seja, desde a criação até o PO.
nPullsLifetimeT	Quantidade de pull requests realizados durante o tempo de vida da issue, ou seja, desde a criação até o PO.
nCommitsLifetimeT	Quantidade de commits realizados durante o tempo de vida da issue, ou seja, desde a criação até o PO.

6.2.3 Atributos temporais

Os atributos temporais estão relacionados com datas de criação das atividades. Tudo que acontece no sistema do GitHub possui uma data indicando quando o evento ocorreu e informações relevantes podem ser extraídas sobre estas datas. Após as *issues* serem reportadas, podem ocorrer eventos relacionados ou serem postados comentários em sua discussão. Todavia, esses comentários e eventos podem demorar para acontecer, indicando que a *issue* foi ignorada por um tempo ou permanentemente. Também pode ocorrer de muitas atividades relacionadas à *issue* acontecerem logo quando ela é reportada. A ocorrência de muitos eventos e comentários em uma *issue* em um curto intervalo de tempo aponta que os desenvolvedores estão empenhados em resolvê-la, e, provavelmente, tem-se um impacto no tempo de resolução.

Kikas, Dumas e Pfahl (2016) utiliza atributos dinâmicos mas nenhum deles se refere ao aspecto temporal. Todos os atributos temporais são propostos por este trabalho

e listados na Tabela 4, juntamente com suas descrições. O atributo "weekday"informa o dia da semana em que a *issue* foi criada, levando em consideração a hipótese que *issues* criadas no final de semana podem demorar até o próximo dia útil para serem analisadas e resolvidas. Algumas *issues*, mesmo sendo relevantes, levam um tempo para a equipe começar a trabalhar em sua resolução, devido a limitações de recursos humanos e a necessidade de priorizar outras *issues*. Esse tempo de inatividade é capturado pelos atributos "cmtActivationTimeT"e "eventActivationTime". *Issues* podem ter picos de atividade e posteriormente tornarem-se inativas. O tempo desde a última atividade ocorrida é informado pelos atributos "cmtDowntimeT"e "eventDowntimeT". Todos esses atributos são justificáveis ao se considerar que provavelmente uma *issue* inativa demanda mais tempo para ser resolvida do que uma ativa.

A Figura 3 exemplifica os intervalos entre comentários utilizados para criação dos atributos. Como no exemplo, os intervalos de tempo entre a criação dos eventos e comentários trazem informações sobre a frequência de atividades em uma *issue*. As atividades podem ocorrer esporadicamente, com intervalos bem espaçados; ou de forma frequente, com pequenos intervalos entre elas. Os intervalos são medidos por meio da diferença em segundos entre as datas de criação. Sobre esses intervalos são calculados atributos que consistem em medidas estatísticas relacionadas à sua distribuição, tanto para os intervalos entre eventos, quanto para os intervalos entre comentários (Tabela 4). Uma vez que as atividades são dinâmicas, é possível que se aumente ou diminua o tamanho dos intervalos entre elas, à medida que o tempo passa e são considerados pontos de observação maiores. Por exemplo, em casos que a *issue* é ignorada por um tempo e, posteriormente, desencadeia-se uma série de atividades sobre ela, após um desenvolvedor postar um comentário.

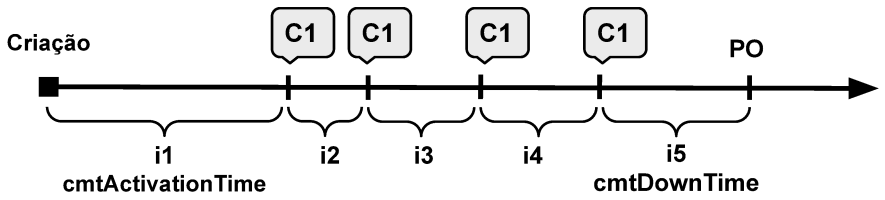

Figura 3 – Intervalos entre os comentários de uma *issue*, "C"representa os comentários e "i"os intervalos

6.2.4 Atributos textuais

Atributos textuais são informações extraídas do texto presente nas descrições e comentários das *issues*. Na maioria dos casos, o texto é a única informação fornecida pe-

Tabela 4 – Atributos temporais calculados utilizando as datas, todos intervalos são medidos em segundos.

Atributo	Descrição
Issue	
weekday	Dia da semana que a issue foi criada
Comentários da issue	
cmtActivationTimeT	Tempo desde a criação até o primeiro comentário ser postado, considerando o PO como limite
cmtDowntimeT	Tempo desde o ultimo comentário postado até o PO
minCmtIntervalT	Menor intervalo de tempo entre os comentários postados antes do PO
maxCmtInterval	Maior intervalo de tempo entre os comentários postados antes do PO
stdCmtInterval	Desvio padrão dos intervalos de tempo entre os comentários postados antes do PO
meanCmtInterval	Média dos intervalos de tempo entre os comentários postados antes do PO
q1CmtInterval	Primeiro quartil dos intervalos de tempo entre os comentários postados antes do PO
q3CmtInterval	Segundo quartil dos intervalos de tempo entre os comentários postados antes do PO
Eventos da issue	
eventActivationTime	Tempo desde a criação até o primeiro evento ocorrer, considerando o PO como limite
eventDowntime	Tempo desde último evento ocorrido até o PO
minEventInterval	Menor intervalo de tempo entre os eventos ocorridos antes do PO
maxEventInterval	Maior intervalo de tempo entre os eventos ocorridos antes do PO
stdEventInterval	Desvio padrão dos intervalos de tempo entre os eventos ocorridos antes do PO
meanEventInterval	Média dos intervalos de tempo entre os eventos ocorridos antes do PO
q1EventInterval	Primeiro quartil dos intervalos de tempo entre os eventos ocorridos antes do PO
q2EventInterval	Segundo quartil dos intervalos de tempo entre os eventos ocorridos antes do PO

los usuários ao se criar uma *issue*. Embora o sistema de *issue tracking* do GitHub possua campos como labels, milestone, usuário atribuído, esses campos quase nunca são preenchidos. Isso pode acontecer porque existem casos em que o usuário não tem conhecimento sobre essas informações mas quer relatar uma questão. Todavia, é possível identificar informações similares as contidas nesses campos, a partir do texto, e essa é a importância dos atributos textuais.

Boa parte dos trabalhos relacionados não enfatizam a extração de atributos textuais, limitando-se a um ou dois atributos ou até mesmo não utilizando o texto como fonte de informação. Todos os atributos textuais são apresentados na Tabela 5, sendo 2 deles reaproveitados do trabalho de Kikas, Dumas e Pfahl (2016) e outros 11 propostos neste trabalho.

A Figura 4 exemplifica o conceito presente nos atributos relativos à quantidade de menções no texto. Nota-se que o autor da *issue* em exemplo não atribuiu ela a alguém,

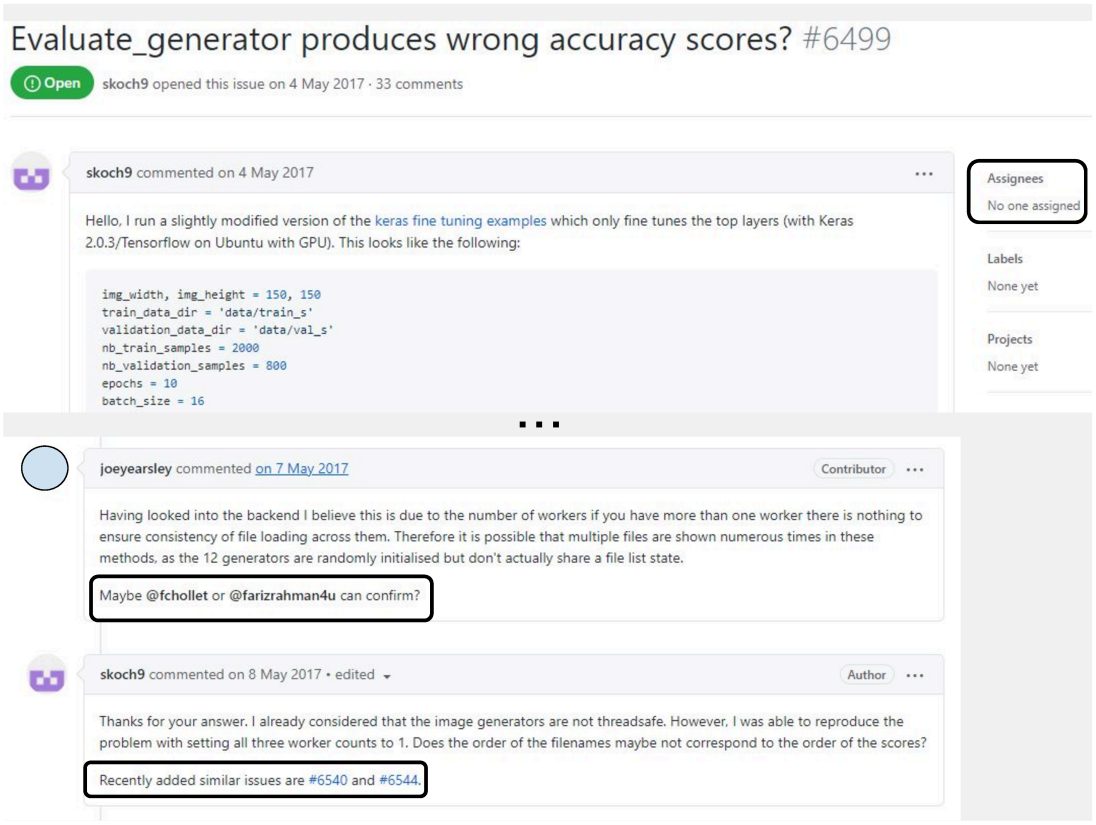

Figura 4 – Recortes de uma *issue* com menções a desenvolvedores e outras *issues*

mas um desenvolvedor fez um comentário mencionando pessoas que provavelmente têm conhecimento sobre a questão a ser resolvida. As menções podem ser realizadas em qualquer texto por meio do padrão nome para usuários, e #número para as *issues*. Existem casos em que outra *issue* é mencionada para indicar que é uma duplicata da questão relatada ou está relacionada, como também pode ser visto na Figura 4.

Outro recurso disponível é a inclusão de trechos de código nos comentários. Esses trechos são destacados do resto do texto pela interface do Github. Todavia, ao extrair os textos presentes em uma *issue* por meio da GitHub API não existe uma separação dos trechos de código e é impossibilitada a distinção desses trechos. Por essa razão os atributos "descWordRate", "FirstCmtsWordRateT"e "LastCmtsWordRateT"são criados, numa tentativa de informar a proporção do texto que é trecho de código por meio da proporção de palavras. Uma vez que em trechos de código os termos são frequentemente compostos por palavras concatenadas e caracteres especiais e não são reconhecidos como palavras.

Existe a opção de incluir labels nas *issues*, categorizando-as de acordo com o tipo da questão (erro, aprimoramento, entre outras). Contudo, são poucas as *issues* que possuem labels. Nesse sentido, a técnica *latent direct allocation* (LDA) é aplicada sobre os textos a fim de descobrir os tópicos presentes neles. O LDA identifica os tópicos de forma automatizada e calcula a probabilidade do texto pertencer a cada tópico, sendo necessário

definir a quantidade de tópicos desejada. O atributo "20LDATopicsOfDesc"fornece 20 valores para os modelos, um para cada tópico, mas foi agrupado em uma só linha da tabela para fins de simplificação.

Tabela 5 – Atributos textuais

Atributo	Descrição
Issues	
issueCleanedBodyLen	Kikas et al.: Comprimento do título e descrição juntos, após a remoção de Stopwords
meanCommentSizeT	Kikas et al.: Tamanho médio dos comentários postados até o PO, após a remoção de Stopwords
nIssueMentionsTitle	Quantidade de menções a outras issues no título
nDevsMentionsDesc	Quantidade de menções a desenvolvedores na descrição
descWordRate	Proporção de termos que são palavras na descrição
20 LDATopicsOfDesc	Probabilidade do texto do título e descrição pertencer a cada um dos 20 tópicos gerados com LDA
Comentários das issues	
nIssueMentionsInFirstCmtsT	Quantidades de menções a outras issues nos dois primeiros comentários
nIssueMentionsInLastCmtsT	Quantidades de menções a outras issues nos dois últimos comentários
nDevsMentionsInFirstCmtsT	Quantidades de menções a desenvolvedores nos dois primeiros comentários
nDevsMentionsInLastCmtsT	Quantidades de menções a desenvolvedores nos dois últimos comentários
FirstCmtsWordRateT	Proporção de termos que são palavras nos dois primeiros comentários
LastCmtsWordRateT	Proporção de termos que são palavras nos dois últimos comentários
10 LDATopicsOfLastCmtsT	Probabilidade do texto dos últimos comentários pertencer a cada um dos 10 tópicos gerados com LDA

6.3 Treinamento dos Modelos

Tendo as características a serem fornecidas aos modelos definidas, torna-se viável o treinamento. O processo de treinamento permite que os modelos identifiquem padrões sobre os dados, utilizando as informações contidas nos atributos e algoritmos de aprendizado de máquina. Após o treinamento, os padrões aprendidos são usados pelos modelos para classificar as novas *issues*.

Os modelos podem se ajustar excessivamente durante a fase de treinamento, obtendo um bom desempenho ao prever as classes dos dados de treino, mas cometendo muitos erros ao prever a classe de novos dados. Sendo assim, é necessário utilizar dados não vistos anteriormente para realizar uma avaliação válida dos modelos. Por essa razão, os dados são divididos em conjuntos para treino e teste, conforme descrito nas próximas subseções.

6.3.1 Divisão do conjunto de treino e teste

Uma vez que são considerados aspectos temporais sobre os dados, é importante não utilizar dados futuros para prever o passado, pois uma situação desse tipo seria impossível no mundo real. Ao aplicar o modelo, estarão disponíveis para treino somente dados do passado e presente e os dados futuros não são conhecidos. Tendo em vista essa restrição, a divisão do conjunto de treino e teste deve considerar as datas em que as *issues* foram criadas. Desse modo, os dados são divididos de acordo com uma data. A data escolhida para divisão é 30 de janeiro de 2017, sendo que todas as *issues* criadas antes desta data fazem parte do conjunto de treino e as criadas depois fazem parte do conjunto de teste. A escolha da data para divisão foi realizada de forma que sobrassem dados suficientes para teste dos modelos com o ponto de observação de 180 dias, devido ao fato dele adicionar restrições aos conjuntos e torná-los menores. Os conjuntos se diferem de acordo com o ponto de observação, pois somente *issues* que duraram até o PO podem estar presentes. O horizonte de tempo também impõe restrições para as *issues* próximas à data limite dos dados, sendo que *issues* abertas devem durar até o HT para que seja possível afirmar que não fecharam antes dele e a atribuir a classe 0 à elas.

Kikas, Dumas e Pfahl (2016) dividem os dados da mesma forma, mas utilizam datas diferentes, pois seus dados são mais antigos. A forma de divisão é mantida nesta pesquisa, pois faz sentido e permite uma melhor comparação entre os modelos propostos e os de referência. A restrição temporal impede a aplicação de técnicas para validação tradicionais como *cross-validation* (BROWNE, 2000) e, em virtude desta limitação, um conjunto de validação é extraído, a fim de permitir a configuração dos algoritmos de aprendizado.

6.3.2 Divisão do conjunto de validação

Boa parte dos algoritmos de aprendizado possuem parâmetros que podem ser alterados para fins de adaptação ao problema. Em especial as redes neurais, cujo tamanho e arquitetura podem variar. Ao realizar a configuração dos algoritmos de aprendizado de máquina, também deve-se utilizar dados não vistos no processo de treinamento, uma vez que modelos com configurações diferentes são comparados com o intuito de se escolher o que apresenta o melhor desempenho. A escolha com base nos resultados obtidos a partir do conjunto de treino favorece um super-ajuste do modelo. O mesmo ocorre caso seja baseada no conjunto de teste. Portanto, demanda-se pela separação de mais um conjunto, dedicado à validação das configurações realizadas.

Como os dados para teste não devem ser utilizados para outros fins além da avaliação dos modelos, o conjunto de validação é extraído do conjunto de treino. Novamente, o aspecto temporal dos dados deve ser considerado ao se realizar a divisão. Para definir o critério de separação, foram realizados testes utilizando as divisões por data e por projeto,

sendo que esta última possibilitou uma validação das configurações mais assertiva. Ao se usar uma data para divisão, projetos criados após tal data não fazem parte do conjunto de treino mas são incluídos no conjunto de validação. Isso faz com que as *issues* desses projetos se comportem como ruído e atrapalhem a escolha da melhor configuração. Sendo assim, a separação por projeto se mostrou mais adequada, sendo selecionadas 20% das *issues* mais recentes de cada projeto para validação e os 80% restantes para treino.

A utilização do conjunto para validação, nesta pesquisa, permitiu a configuração e a comparação dos algoritmos de aprendizado, destacando-se em relação ao trabalho de referência (KIKAS; DUMAS; PFAHL, 2016) que escolhe a configuração de forma arbitrária, sem validação.

6.3.3 Escolha do classificador

Prever se a *issue* irá fechar ou não antes do horizonte de tempo é um problema de classificação binária. Para resolver esse tipo de problema, os algoritmos de aprendizado de máquina são utilizados como classificadores, pois classificam as *issues* em classes positivas (irá fechar) ou negativas (não irá fechar). Existem diversos algoritmos presentes na literatura para esse fim e o desempenho deles varia de acordo com o problema em que são aplicados. Assim, é necessário compará-los a fim de se identificar qual se adapta melhor ao contexto em questão.

Kikas, Dumas e Pfahl (2016) utilizam Random Forests, definindo arbitrariamente a seguinte configuração: 1000 árvores de decisão com altura máxima de 5. Os autores justificam a escolha por causa do extenso trabalho para encontrar os parâmetros adequados para cada um dos 28 modelos, devido ao tempo demandado para treinar as Random Forests. Neste trabalho, foi comprovado o extenso tempo demandado ao se investigar configurações diferentes. Sendo assim, tal investigação foi descartada devido à existência de algoritmos mais rápidos, mesmo ao lidar com grandes volumes de dados. Além disso, Random Forests não oferecem recursos para tratar dados desbalanceados (mais instâncias de uma classe do que da outra), sendo necessário excluir instâncias de uma das classes em alguns casos, a fim de manter as duas na mesma proporção.

Diante das limitações observadas com o uso de Ramdom Forests, este trabalho investigou o uso de Redes Neurais Artificiais (RNAs) como alternativa para o classificador. As RNAs possuem a vantagem de utilizar GPUs para acelerar o processo de aprendizado. Além disso, elas lidam melhor com ruído e problemas complexos e são consideradas uma boa escolha para resolução de problemas de estimativa de esforço (SRINIVASAN; FISHER, 1995). Elas também permitem a utilização de pesos para lidar com o desbalanceamento de classes. Admitindo a Rede Neural como classificador, é necessário estabelecer a quantidade de camadas da rede, e para cada camada, a quantidade de neurônios, sendo a seguinte arquitetura definida neste trabalho:

- **Camada de entrada**
- **Camada escondida**: 240 neurônios, com dropout
- **Camada escondida**: 110 neurônios, com dropout
- **Camada saída**: 1 neurônio.

Foram avaliadas redes com menos neurônios e camadas mas elas não identificavam padrões robustos sobre os dados. Redes maiores obtiveram um bom desempenho para prever a classe negativa (quando a *issue* não irá fechar) mas não eram boas ao prever a classe positiva. É provável que isso esteja relacionado ao desbalanceamento das classes. Para contornar esse problema, pesos são definidos para cada classe ao inicializar a rede. O desempenho das diferentes arquiteturas foi avaliado de forma geral, considerando todos os 28 modelos propostos.

A técnica de *dropout* evita super-ajuste da rede ao desativar alguns neurônios aleatórios a cada iteração. Nos testes realizados, a adição de *dropout* com a taxa de exclusão de 0.3 impediu que a rede deteriorasse após algumas iterações, superajustando para o conjunto de treino. Outra técnica bastante utilizada e empregada neste trabalho para evitar superajuste é a parada antecipada (*earling stopping*). Com essa técnica, o treinamento é interrompido quando a rede para de melhorar as classificações do conjunto de validação. São consideradas 100 iterações de paciência, ou seja, o treinamento é interrompido quando a perda não diminui após 100 iterações. São utilizadas as funções de ativação Sigmoid[1] para a camada de saída e ReLU[2] para as restantes. A taxa de aprendizado considerada é de 0.001.

6.4 Considerações finais

A etapa de desenvolvimento evolveu processos para criação de modelos de aprendizado de máquina capazes de estimar o tempo de resolução das *issues*. O trabalho de Kikas, Dumas e Pfahl (2016) foi escolhido como fase inicial e modelos de referência foram desenvolvidos, mantendo a mesma modelagem. Foram criados 28 modelos variando os pontos de observação e horizontes de tempo. As melhorias são realizadas visando aprimorar os modelos e a qualidade das estimativas, explorando conceitos e técnicas não exploradas em outros trabalhos. Boa parte das contribuições se encontram no processo de engenharia de atributos. Esse processo mostra-se importante pela escassez de dados referentes às *issue* no GitHub e destacam-se os atributos temporais e textuais propostos, além dos novos atributos de outras categorias. A descrição e justificativa dos atributos propostos são definidas a fim de facilitar sua utilização em outras pesquisas. Também merece destaque a

[1] https://en.wikipedia.org/wiki/Sigmoid_function
[2] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

escolha de um algoritmo de aprendizado mais robusto, usando redes neurais, bem como o processo de configuração e definição da arquitetura utilizando o conjunto de validação.

Com os atributos e algoritmos prontos e os conjuntos de treino e teste estabelecidos, os 28 modelos são treinados. Por meio do conjunto de teste, é possível avaliar o desempenho dos modelos, bem como, as melhorias que aprimoramentos trouxeram para a qualidade das estimativas.

7 Resultados e Discussão

É importante avaliar o desempenho dos modelos desenvolvidos, a fim de se obter uma noção sobre a qualidade das estimativas realizadas. Com esse propósito, as métricas acurácia e precisão são calculadas com base nas definições presentes no Capítulo 2. O objetivo desde capítulo é apresentar os resultados obtidos com os modelos propostos, comparando-os com os modelos de referência (KIKAS; DUMAS; PFAHL, 2016). Também será investigada importância dos atributos propostos, avaliando se as categorias de atributos exploradas neste trabalho realmente contribuem para o poder preditivo dos modelos. Para isso, as seguintes hipóteses (Capítulo 1) serão investigadas neste capítulo.

1. H1: É possível estimar com acurácia e precisão o tempo de resolução de *issues* no GitHub, considerando-se a não estruturação das informações.
2. H2: A adição de atributos textuais resulta em melhorias na qualidade das estimativas.
3. H3: A adição de atributos temporais resulta em melhorias na qualidade das estimativas.
4. H4: O uso de algoritmos de aprendizado, como redes neurais, mostra-se mais adequado para resolver o problema.

A Seção 7.1 apresenta os resultados gerais dos modelos propostos e discute a hipótese H1 e H4. As discussões sobre a importância dos atributos para os diferentes modelos são abordadas na Seção 7.2, considerando tando as categorias em específico quando os atributos em geral, a fim de responder as hipóteses H2 e H3.

7.1 Avaliação dos modelos

As métricas acurácia, precisão, revocação e F-score são calculadas para avaliar os modelos, sendo considerados todos os 28 modelos, variando-se o ponto de observação e o horizonte de tempo. Os testes abrangem os modelos de referência, baseados no trabalho de Kikas, Dumas e Pfahl (2016), com os mesmos atributos, classificador (Random Forest) e configurações definidas pelos autores. Os modelos também são testados utilizando-se *Multi Layer Perceptron* (MLP) como classificador, mas ainda considerando os mesmos atributos de Kikas, Dumas e Pfahl (2016) com o propósito de se avaliar o impacto do uso de redes neurais. Por fim, avalia-se os modelos finais, considerando as melhorias referentes à troca

Tabela 6 – Comparação entre os modelos para os diferentes POs e HTs

	Acurácia			**Precisão**			**Revocação**			**F-score**		
Modelo	**Kikas et al.**	**MLP kikas et.al**	**MLP Novos Atb**	**Kikas et al.**	**MLP kikas et.al**	**MLP Novos Atb**	**Kikas et al.**	**MLP kikas et.al**	**MLP Novos Atb**	**Kikas et al.**	**MLP kikas et.al**	**MLP Novos Atb**
P0H1	0,596	**0,682**	0,651	0,317	**0,368**	0,352	**0,692**	0,582	0,666	0,435	0,451	**0,461**
P0H7	0,612	**0,664**	0,658	0,498	**0,561**	0,546	**0,707**	0,597	0,676	0,584	0,579	**0,604**
P0H14	0,619	0,659	**0,664**	0,562	**0,625**	0,615	**0,711**	0,616	0,684	0,628	0,620	**0,648**
P0H30	0,627	0,653	**0,668**	0,628	0,675	**0,679**	**0,714**	0,658	0,700	0,668	0,666	**0,690**
P0H90	0,655	0,663	**0,688**	0,726	**0,752**	0,751	0,743	0,710	**0,770**	0,734	0,730	**0,760**
P0H180	0,675	0,680	**0,708**	0,792	0,810	**0,817**	0,748	0,730	**0,770**	0,769	0,768	**0,793**
P0H365	0,708	0,713	**0,737**	0,884	0,895	**0,900**	0,751	0,745	**0,772**	0,812	0,813	**0,831**
P1H7	0,545	**0,669**	0,663	0,275	0,335	**0,342**	**0,733**	0,610	0,657	0,400	0,433	**0,450**
P1H14	0,563	0,649	**0,664**	0,372	0,432	**0,448**	**0,727**	0,641	0,662	0,492	0,516	**0,534**
P1H30	0,584	0,643	**0,669**	0,476	0,530	**0,558**	**0,729**	0,676	0,695	0,576	0,595	**0,619**
P1H90	0,617	0,655	**0,696**	0,618	0,673	**0,703**	0,728	0,681	**0,747**	0,669	0,677	**0,711**
P1H180	0,639	0,668	**0,716**	0,711	0,755	**0,783**	0,724	0,705	**0,763**	0,717	0,729	**0,773**
P1H365	0,685	0,706	**0,763**	0,836	0,859	**0,879**	0,742	0,745	**0,807**	0,786	0,798	**0,841**
P7H14	0,599	**0,683**	0,679	0,157	0,187	**0,187**	**0,648**	0,600	0,616	0,253	0,286	**0,287**
P7H30	0,599	0,665	**0,676**	0,310	0,357	**0,374**	0,652	0,632	**0,677**	0,420	0,456	**0,482**
P7H90	0,615	0,658	**0,699**	0,513	0,558	**0,603**	0,625	0,667	**0,707**	0,563	0,607	**0,651**
P7H180	0,629	0,656	**0,715**	0,642	0,656	**0,715**	0,648	0,711	**0,756**	0,645	0,683	**0,734**
P7H365	0,649	0,699	**0,746**	0,805	0,802	**0,859**	0,657	0,755	**0,762**	0,723	0,778	**0,808**
P14H30	0,604	0,650	**0,686**	0,194	0,217	**0,236**	0,647	**0,651**	0,632	0,298	0,326	**0,343**
P14H90	0,609	0,656	**0,707**	0,427	0,473	**0,533**	0,646	0,664	**0,705**	0,514	0,553	**0,607**
P14H180	0,634	0,648	**0,719**	0,583	0,593	**0,668**	0,676	0,717	**0,753**	0,626	0,649	**0,708**
P14H365	0,663	0,686	**0,755**	0,803	0,780	**0,826**	0,634	0,715	**0,787**	0,708	0,746	**0,806**
P30H90	0,625	0,659	**0,726**	0,308	0,336	**0,404**	0,629	**0,641**	0,638	0,413	0,441	**0,494**
P30H180	0,683	0,648	**0,727**	0,544	0,502	**0,594**	0,644	0,672	**0,714**	0,589	0,574	**0,649**
P30H365	0,679	0,672	**0,750**	0,744	0,700	**0,795**	0,655	0,732	**0,748**	0,697	0,716	**0,771**
P90H180	0,687	0,682	**0,763**	0,284	0,278	**0,353**	**0,665**	0,654	0,631	0,398	0,390	**0,453**
P90H365	0,702	0,686	**0,769**	0,591	0,576	**0,678**	0,665	0,611	**0,730**	0,626	0,593	**0,703**
P180H365	0,714	0,695	**0,796**	0,379	0,359	**0,491**	**0,685**	0,679	0,661	0,488	0,470	**0,563**

do classificador e à adição dos novos atributos propostos. Os resultados são apresentados na Tabela 6, e para cada métrica, os melhores valores são destacados em negrito.

Observa-se que grande parte dos melhores resultados são obtidos pelos modelos propostos, como pode ser visto nas colunas "MLP Novos Atb" da Tabela 6. A acurácia dos modelos propostos varia de 65% a 79% e melhora consideravelmente em relação ao modelo de referência, com um aumento de 7 pontos em média, chegando a 10 pontos a mais em modelos como P1H7, P1H14 e P30H90. A precisão varia de 18% a 90%, com um aumento em torno de 6 pontos. Contudo a revocação é a única métrica que apresenta piora em alguns modelos, principalmente aqueles com o ponto de observação igual a 0. Mesmo assim, existe uma pequena melhora na maioria dos modelos e os valores de revocação variam de 61% a 78%. Além disso, o ganho na precisão sempre compensa a perda na revocação, pois os valores de *F-score* são sempre melhores. O *F-score* varia de 28% a 84% e aumenta em média 5 pontos em relação ao modelo de referência.

Para cada ponto de observação, os menores resultados se referem aos modelos com os primeiros horizontes considerados, como o P0H1, P1H7, P7H14, P14H30 e assim por diante. Apesar dos modelos propostos apresentarem melhorias, eles ainda possuem uma baixa precisão para a classe positiva (quando a *issue* irá fechar). Contudo, obtém-se uma

boa acurácia e tais modelos são bons ao prever a classe negativa (quando a *issue* não fechará). Esses resultados provavelmente estão relacionados ao maior desbalanceamento das classes. Por exemplo, no caso do P0H1, é pouco provável que uma *issue* seja fechada em apenas 1 dia, totalizando somente 18% das *issues*, assim como no P7H14, onde poucas *issues* que duraram 7 dias são resolvidas em 14 dias (9% do total).

Os resultados são satisfatórios e as melhorias fazem com que os modelos propostos superem a referência. Em geral, a adição das redes neurais por si só já provoca uma melhoria nos resultados, indicando que elas são mais apropriadas para prever o tempo de resolução das *issues* do que as *Random Forests* no contexto desta pesquisa. A adição dos novos atributos também melhora os resultados, com algumas exceções em relação à acurácia e à precisão quando o horizonte de tempo é baixo, onde os modelos sem esses novos atributos e somente com a adição da rede neural apresentam o melhor desempenho. Observou-se que os maiores ganhos ao se adicionar atributos se concentram nos pontos de observação maiores que 0, provavelmente por conta da importância dos novos atributos dinâmicos.

7.2 Avaliação do impacto dos atributos nos modelos

Ao prever as classes das *issues* alguns atributos fornecem informações mais relevantes do que outros, ao estarem mais relacionados com o tempo de resolução. Com a intenção de investigar o impacto dos novos atributos no desempenho dos modelos, serão apresentados resultados de testes, nesta seção, para avaliar o impacto após a adição dos atributos. Sendo assim, as seguintes hipóteses de pesquisa foram investigadas:

- H2: A adição de atributos textuais resulta em melhorias na qualidade das estimativas.
- H3: A adição de atributos temporais resulta em melhorias na qualidade das estimativas.

7.2.1 Avaliação dos atributos textuais

Para se avaliar o impacto do uso dos atributos textuais, foram realizados testes nos modelos antes e depois da adição destes atributos. Todos os modelos considerados utilizam MLP como classificador, com a mesma configuração descrita no Capítulo 6. Além disso, eles são treinados e testados utilizando o mesmo conjunto de dados. A Tabela 7 apresenta os resultados obtidos, onde a primeira coluna considera os modelos somente com os atributos de Kikas, Dumas e Pfahl (2016). Tais atributos estão presentes em todos modelos das outras colunas, tornando a adição dos atributos a única diferença entre eles. A segunda coluna apresenta os resultados após a adição do LDA (Seção 6.2). Na terceira,

por sua vez, são adicionados os outros atributos textuais, mas sem o LDA. O LDA é isolado dos outros atributos por acrescentar muitos valores, sendo 20 tópicos referentes ao título e à descrição e 10 referentes aos comentários.

Tabela 7 – Comparação dos modelos após a adição dos atributos textuais

	Acurácia			**Precisão**			**Revocação**			**F-score**		
Modelo	**MLP kikas et.al**	**Add LDA**	**Add Atb text**	**MLP kikas et.al**	**Add LDA**	**Add Atb text**	**MLP kikas et.al**	**Add LDA**	**Add Atb text**	**MLP kikas et.al**	**Add LDA**	**Add Atb text**
P0H1	0,682	0,659	**0,687**	0,368	0,356	**0,3738**	0,582	**0,645**	0,584	0,451	**0,459**	0,456
P0H7	**0,664**	**0,664**	0,662	**0,561**	0,555	0,555	0,597	**0,646**	0,626	0,579	**0,597**	0,588
P0H14	0,659	**0,663**	0,662	0,625	0,621	**0,633**	0,616	**0,654**	0,595	0,620	**0,637**	0,614
P0H30	0,653	**0,662**	0,658	0,675	**0,683**	0,678	0,658	**0,670**	0,667	0,666	**0,677**	0,673
P0H90	0,663	0,666	**0,667**	0,752	**0,756**	**0,756**	**0,710**	0,708	0,709	0,730	0,731	**0,732**
P0H180	**0,680**	0,669	0,675	0,810	**0,820**	0,813	**0,730**	0,697	0,717	**0,768**	0,754	0,762
P0H365	**0,713**	0,700	0,707	0,895	**0,897**	0,895	**0,745**	0,726	0,739	**0,813**	0,803	0,809
P1H7	0,669	**0,725**	0,652	0,335	**0,349**	0,329	0,610	0,377	**0,661**	0,433	0,362	**0,440**
P1H14	0,649	0,651	**0,662**	0,432	0,433	**0,444**	0,641	**0,644**	0,637	0,516	0,518	**0,523**
P1H30	0,643	0,653	**0,661**	0,530	0,543	**0,554**	**0,676**	0,651	0,637	**0,595**	0,592	0,592
P1H90	0,655	0,655	**0,660**	0,673	0,678	**0,682**	**0,681**	0,668	0,672	0,677	0,673	**0,678**
P1H180	0,668	0,663	**0,675**	**0,755**	0,752	0,753	0,705	0,699	**0,724**	0,729	0,725	**0,738**
P1H365	0,706	0,695	**0,714**	**0,859**	**0,859**	**0,859**	0,745	0,729	**0,757**	0,798	0,788	**0,805**
P7H14	**0,683**	0,669	0,658	**0,187**	0,178	0,179	0,600	0,600	**0,634**	**0,286**	0,275	0,280
P7H30	**0,665**	0,664	0,664	0,357	**0,357**	0,352	0,632	**0,640**	0,611	0,456	**0,459**	0,447
P7H90	**0,658**	0,648	**0,658**	**0,558**	0,546	0,555	0,667	0,668	**0,6887**	0,607	0,601	**0,615**
P7H180	0,656	0,652	**0,664**	0,656	0,660	**0,669**	**0,711**	0,685	0,700	0,683	0,672	**0,684**
P7H365	0,699	0,690	**0,705**	0,802	0,807	**0,810**	**0,755**	0,732	0,753	0,778	0,767	**0,781**
P14H30	0,650	0,641	**0,665**	0,217	0,205	**0,221**	**0,651**	0,614	0,626	0,326	0,307	**0,327**
P14H90	0,656	0,653	**0,661**	0,473	0,469	**0,479**	0,664	0,630	**0,671**	0,553	0,537	**0,559**
P14H180	0,648	0,648	**0,651**	0,593	0,593	**0,595**	0,717	0,711	**0,720**	0,649	0,646	**0,652**
P14H365	0,686	0,684	**0,694**	**0,780**	0,764	0,778	0,715	**0,740**	0,735	0,746	0,752	**0,756**
P30H90	0,659	**0,666**	0,655	**0,336**	**0,336**	0,334	0,641	0,607	**0,648**	**0,441**	0,433	**0,441**
P30H180	**0,648**	0,645	0,641	**0,502**	0,499	0,494	0,672	0,636	**0,700**	0,574	0,559	**0,580**
P30H365	0,672	0,670	**0,675**	0,700	0,696	**0,701**	0,732	**0,736**	**0,736**	0,716	0,716	**0,718**
P90H180	0,682	**0,684**	0,682	0,278	0,274	**0,280**	0,654	0,627	**0,668**	0,390	0,382	**0,395**
P90H365	**0,686**	0,679	0,670	**0,576**	0,567	0,547	0,611	0,610	**0,694**	0,593	0,588	**0,612**
P180H365	0,695	0,657	**0,708**	0,359	0,328	**0,372**	0,679	**0,688**	0,684	0,470	0,444	**0,482**

7.2.2 Avaliação dos atributos temporais

A importância dos atributos temporais é investigada avaliando-se os resultados após a adição destes nos modelos. Também utiliza-se o mesmo classificador e os conjuntos de dados em todos os modelos. A Tabela 8 apresenta os resultados obtidos, onde a primeira coluna considera somente os modelos com os atributos propostos por Kikas, Dumas e Pfahl (2016) e a segunda, os modelos após a adição dos atributos temporais.

Em geral, os atributos temporais melhoram os resultados à medida que são considerados pontos de observação maiores, mas não há melhorias quando o ponto de observação é zero. Isso ocorre, provavelmente, em virtude da maioria dos atributos dessa categoria serem dinâmicos. Observa-se uma melhora expressiva nas métricas precisão e acurácia, com destaque nos modelos com PO igual ou acima de 14 dias. Isso ocorre pois há um intervalo de tempo maior e são considerados mais eventos ao calcular os atributos. Contudo, a revocação piora em alguns casos, mas no F-score em geral não há diferenças significativas.

Tabela 8 – Comparação dos modelos após a adição dos atributos temporais

	Acurácia		**Precisão**		**Revocação**		**F-score**	
Modelo	**MLP kikas et.al**	**Add Atb Temporal**	**MLP kikas et.al**	**Add Atb Temporal**	**MLP kikas et.al**	**Add Atb Temporal**	**MLP kikas et.al**	**Add Atb Temporal**
P0H1	0,682	0,682	0,368	0,368	**0,582**	0,581	**0,451**	0,450
P0H7	**0,664**	0,645	0,561	**0,562**	**0,597**	0,589	**0,579**	0,575
P0H14	**0,659**	0,656	**0,625**	0,621	**0,616**	0,615	**0,620**	0,618
P0H30	0,653	**0,654**	**0,675**	0,672	0,658	**0,671**	0,666	**0,671**
P0H90	0,663	**0,666**	0,752	0,752	0,710	**0,715**	0,730	**0,733**
P0H180	**0,680**	0,678	**0,810**	0,808	**0,730**	0,729	**0,768**	0,767
P0H365	**0,713**	0,707	**0,895**	0,893	**0,745**	0,740	**0,813**	0,809
P1H7	**0,669**	0,653	**0,335**	0,329	0,610	**0,650**	0,433	**0,437**
P1H14	0,649	**0,662**	0,432	**0,444**	**0,641**	0,626	0,516	**0,519**
P1H30	0,643	**0,645**	0,530	**0,532**	0,676	0,676	0,595	0,595
P1H90	0,655	**0,657**	0,673	**0,685**	**0,681**	0,655	**0,677**	0,670
P1H180	0,668	**0,672**	0,755	0,755	0,705	**0,716**	0,729	**0,735**
P1H365	0,706	**0,709**	**0,859**	0,858	0,745	**0,751**	0,798	**0,801**
P7H14	**0,683**	0,682	0,187	**0,186**	0,600	**0,606**	**0,286**	0,285
P7H30	0,665	**0,667**	0,357	**0,361**	0,632	**0,643**	0,456	**0,462**
P7H90	0,658	**0,663**	0,558	**0,561**	0,667	**0,6908**	0,607	**0,619**
P7H180	0,656	**0,672**	0,656	**0,689**	**0,711**	0,674	**0,683**	0,682
P7H365	**0,699**	0,694	0,802	**0,825**	**0,755**	0,713	**0,778**	0,765
P14H30	0,650	**0,671**	0,217	**0,223**	**0,651**	0,619	0,326	**0,328**
P14H90	0,656	**0,680**	0,473	**0,500**	**0,664**	0,620	0,553	0,553
P14H180	0,648	**0,661**	0,593	**0,606**	0,717	0,717	0,649	**0,657**
P14H365	0,686	**0,694**	0,780	**0,789**	0,715	**0,720**	0,746	**0,753**
P30H90	0,659	**0,693**	0,336	**0,361**	**0,641**	0,603	0,441	**0,452**
P30H180	0,648	**0,690**	0,502	**0,556**	**0,672**	0,613	0,574	**0,583**
P30H365	0,672	**0,688**	0,700	**0,734**	**0,732**	0,699	0,716	0,716
P90H180	0,682	**0,761**	0,278	**0,334**	**0,654**	0,537	0,390	**0,412**
P90H365	0,686	**0,717**	0,576	**0,621**	0,611	**0,624**	0,593	**0,623**
P180H365	0,695	**0,738**	0,359	**0,404**	**0,679**	0,667	0,470	**0,503**

Em resumo, a adição dos atributos temporais melhora a acurácia dos modelos PO maior que zero e a precisão é aprimorada ao custo de perdas na revocação.

7.3 Limitações

Esta seção discute algumas limitações encontradas no decorrer do desenvolvimento da pesquisa. Durante o desenvolvimento dos modelos de referência, as descrições presentes no trabalho de Kikas, Dumas e Pfahl (2016) foram mantidas, dentro do possível, mas não há como garantir que os filtros e atributos implementados estejam exatamente iguais. Isso faz com que os modelos propostos, apesar de similares aos da literatura, apresentem suas particularidades, causando um impacto nos resultados obtidos. Além disso, o conjunto de dados utilizado neste trabalho é mais recente do que o presente no trabalho de referência e essa diferença provavelmente interferiu nos resultados obtidos. Outra limitação em relação aos atributos está na definição dos intervalos de tempo considerados. Alguns

consideram dados desde a criação da *issue*, outros 2 semanas ou 3 meses antes da criação. Esses intervalos são definidos intuitivamente, mas estudos futuros podem ser realizados testando-se intervalos diferentes e os impactos causados na qualidade das estimativas. O mesmo vale para a quantidade de tópicos escolhidos no LDA. Neste trabalho, não foi realizada uma busca pela melhor quantidade, devido ao custo de realizar testes para cada ponto de observação causado pela demora para o algoritmo apresentar resultados. Futuramente, podem ser realizadas investigações sobre o impacto nas estimativas ao se escolher diferentes quantidades de tópicos e sobre a qualidade desses tópicos no âmbito das características capturadas sobre os textos.

8 Conclusões e trabalhos futuros

Este trabalho apresentou modelos de estimativa de tempo de resolução de *issues* no GitHub baseados no uso de redes neurais e em atributos textuais e temporais. Ao todo, foram propostos 45 novos atributos e criados 28 modelos, a partir das combinações válidas dos pontos de observação e horizontes de tempo. Foram utilizados dados recentes, totalizando mais de 900 mil *issues* até 2019, e implementados filtros e um extrator de dados textuais, a fim de aumentar a qualidade do conjunto de dados.

O trabalho de Kikas, Dumas e Pfahl (2016) foi escolhido como fase inicial e modelos de referência foram desenvolvidos, mantendo-se a mesma forma de modelagem. As melhorias foram realizadas visando aprimorar os modelos e melhorar a qualidade das estimativas, explorando conceitos e técnicas não exploradas em outros trabalhos. Boa parte delas se encontram no processo de engenharia de atributos, importante devido a escassez de dados referentes às *issue* no GitHub. Destacam-se os atributos temporais e textuais propostos, além dos novos atributos de outras categorias. Outro aprimoramento se refere à escolha de redes neurais como algoritmo de aprendizado, que se mostrou mais adequada, bem como o processo de configuração e definição da arquitetura utilizando o conjunto de validação. A partir dos resultados, as hipóteses de pesquisa foram avaliadas, chegando-se as seguintes conclusões:

1. A partir dos testes realizados, observou-se que os modelos propostos apresentaram um desempenho satisfatório, com destaque para a acurácia. Sendo que a precisão, revocação e F-score são boas na maioria das vezes, mas com algumas exceções, em previsões de curto prazo (PO pequeno). Sendo assim, a hipótese H1 é comprovada.
2. Ao comparar modelos com e sem os atributos textuais não foi identificada uma melhoria significativa nos resultados, refutando a hipótese H2.
3. Ao comparar modelos com e sem atributos temporais foi identificada uma melhoria nos resultados, principalmente quando são considerados pontos de observação maiores, comprovando a hipótese H3.
4. Ao comparar os modelos utilizando Random Forests e Redes Neurais, o desempenho com redes neurais foi superior, tanto na qualidade das estimativas quanto no tempo demandado para treinamento. Dessa forma, a hipótese H4 é parcialmente comprovada, sendo necessários mais testes comparativos com outros algoritmos.

Dentre os trabalhos futuros, destacam-se: o refinamento dos atributos propostos ou desenvolvimento de novos atributos. É possível que melhorias causadas por atributos textuais ocorram ao realizar ajustes na forma como os atributos são definidos. Investigações

sobre o impacto de escolhas como a quantidade de tópicos do LDA e outras configurações podem ser realizadas, a fim de otimizar a definição dos atributos. Também podem ser investigados o desempenho de outros algoritmos de aprendizado além das Random Forests e Redes neurais. Outro trabalho futuro relevante é o desenvolvimento de processos para implantação dos modelos para que seja possível utilizá-los no GitHub.

Referências

AKBARINASAJI, S.; CAGLAYAN, B.; BENER, A. Predicting bug-fixing time: A replication study using an open source software project. **journal of Systems and Software**, Elsevier, v. 136, p. 173–186, 2018.

AL-ZUBAIDI, W. H. A. et al. Multi-objective search-based approach to estimate issue resolution time. In: ACM. **Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering**. [S.l.], 2017. p. 53–62.

ALLAHYARI, M. et al. A brief survey of text mining: Classification, clustering and extraction techniques. **arXiv preprint arXiv:1707.02919**, 2017.

ANBALAGAN, P.; VOUK, M. On predicting the time taken to correct bug reports in open source projects. In: IEEE. **2009 IEEE International Conference on Software Maintenance**. [S.l.], 2009. p. 523–526.

ARDIMENTO, P.; BOFFOLI, N.; MELE, C. A text-based regression approach to predict bug-fix time. In: **Complex Pattern Mining**. [S.l.]: Springer, 2020. p. 63–83.

ARDIMENTO, P.; DINAPOLI, A. Knowledge extraction from on-line open source bug tracking systems to predict bug-fixing time. In: ACM. **Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics**. [S.l.], 2017. p. 7.

BERTRAM, D. et al. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: **Proceedings of the 2010 ACM conference on Computer supported cooperative work**. [S.l.: s.n.], 2010. p. 291–300.

BISSYANDÉ, T. F. et al. Got issues? who cares about it? a large scale investigation of issue trackers from github. In: IEEE. **2013 IEEE 24th international symposium on software reliability engineering (ISSRE)**. [S.l.], 2013. p. 188–197.

BLEI, D. M.; NG, A. Y.; JORDAN, M. I. Latent dirichlet allocation. **the Journal of machine Learning research**, JMLR. org, v. 3, p. 993–1022, 2003.

BOEHM, B.; ABTS, C.; CHULANI, S. Software development cost estimation approaches—a survey. **Annals of software engineering**, Springer, v. 10, n. 1, p. 177–205, 2000.

BOEHM, B. et al. Cost models for future software life cycle processes: Cocomo 2.0. **Annals of software engineering**, Springer, v. 1, n. 1, p. 57–94, 1995.

BROWNE, M. W. Cross-validation methods. **Journal of mathematical psychology**, Elsevier, v. 44, n. 1, p. 108–132, 2000.

BUGZILLA. **Bugzilla Documentation Release 5.0.4**. 2018. <https://buildmedia.readthedocs.org/media/pdf/bugzilla/5.0/bugzilla.pdf>. Acessado em 08 fev. 2021.

CATALDO, M.; HERBSLEB, J. D. Communication networks in geographically distributed software development. In: **Proceedings of the 2008 ACM conference on Computer supported cooperative work**. [S.l.: s.n.], 2008. p. 579–588.

CROWSTON, K. et al. Core and periphery in free/libre and open source software team communications. In: IEEE. **Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)**. [S.l.], 2006. v. 6, p. 118a–118a.

DESTEFANIS, G. et al. On measuring affects of github issues' commenters. In: **Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering**. [S.l.: s.n.], 2018. p. 14–19.

DHASADE, A. B.; VENIGALLA, A. S. M.; CHIMALAKONDA, S. Towards prioritizing github issues. In: **Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference**. [S.l.: s.n.], 2020. p. 1–5.

DÔRES, S. C. N. d. **Um modelo para estimativa de esforço em projetos de reengenharia de software**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2015.

FAN, D. Analysis of critical success factors in it project management. In: IEEE. **Industrial and Information Systems (IIS), 2010 2nd International Conference on**. [S.l.], 2010. v. 2, p. 487–490.

FERNANDEZ-RAMIL, J.; IZQUIERDO-CORTAZAR, D.; MENS, T. What does it take to develop a million lines of open source code? In: SPRINGER. **IFIP International Conference on Open Source Systems**. [S.l.], 2009. p. 170–184.

Gacek, C.; Arief, B. The many meanings of open source. **IEEE Software**, v. 21, n. 1, p. 34–40, 2004.

GIGER, E.; PINZGER, M.; GALL, H. Predicting the fix time of bugs. In: **Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering**. New York, NY, USA: ACM, 2010. (RSSE '10), p. 52–56. ISBN 978-1-60558-974-9. Disponível em: <http://doi.acm.org/10.1145/1808920.1808933>.

GITHUB. **Resources in the REST API**. 2020. <https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting>. Acessado em 08 fev. 2021.

GITHUBGUIDE. **Mastering Issues**. 2020. <https://guides.github.com/features/issues/>. Acessado em 08 fev. 2021.

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>.

HABAYEB, M. et al. On the use of hidden markov model to predict the time to fix bugs. **IEEE Transactions on Software Engineering**, IEEE, v. 44, n. 12, p. 1224–1244, 2017.

HAN, J.; KAMBER, M.; PEI, J. **Data mining concepts and techniques, third edition**. Waltham, Mass.: Morgan Kaufmann Publishers, 2012. Disponível em: <http://www.amazon.de/Data-Mining-Concepts-Techniques-Management/dp/0123814790/ref=tmm_hrd_title_0?ie=UTF8&qid=1366039033&sr=1-1>.

HOOIMEIJER, P.; WEIMER, W. Modeling bug report quality. In: **Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering**. [S.l.: s.n.], 2007. p. 34–43.

JOBLIN, M. et al. Classifying developers into core and peripheral: An empirical study on count and network metrics. In: IEEE. **2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)**. [S.l.], 2017. p. 164–174.

KALLIAMVAKOU, E. et al. Open source-style collaborative development practices in commercial projects using github. In: IEEE. **2015 IEEE/ACM 37th IEEE International Conference on Software Engineering**. [S.l.], 2015. v. 1, p. 574–585.

KALLIAMVAKOU, E. et al. The promises and perils of mining github. In: **Proceedings of the 11th working conference on mining software repositories**. [S.l.: s.n.], 2014. p. 92–101.

KIKAS, R.; DUMAS, M.; PFAHL, D. Issue dynamics in github projects. In: SPRINGER. **International Conference on Product-Focused Software Process Improvement**. [S.l.], 2015. p. 295–310.

KIKAS, R.; DUMAS, M.; PFAHL, D. Using dynamic and contextual features to predict issue lifetime in github projects. In: ACM. **Proceedings of the 13th International Conference on Mining Software Repositories**. [S.l.], 2016. p. 291–302.

LEE, Y. et al. Continual prediction of bug-fix time using deep learning-based activity stream embedding. **IEEE Access**, IEEE, v. 8, p. 10503–10515, 2020.

MADDILA, C.; BANSAL, C.; NAGAPPAN, N. Predicting pull request completion time: a case study on large scale cloud services. In: **Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering**. [S.l.: s.n.], 2019. p. 874–882.

MENDES, E. **Practitioner's knowledge representation: A pathway to improve software effort estimation**. [S.l.]: Springer, 2014. 1-211 p. ISBN 978-3-642-54156-8.

MICROSOFT. **O que é o Processo de Ciência de Dados de Equipe?** 2020. <https://docs.microsoft.com/pt-br/azure/machine-learning/team-data-science-process/overview>. Acessado em 01 mar. 2021.

MITCHELL, T. M. **Machine Learning**. 1. ed. USA: McGraw-Hill, Inc., 1997. ISBN 0070428077.

MOCKUS, A.; FIELDING, R. T.; HERBSLEB, J. D. Two case studies of open source software development: Apache and mozilla. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, ACM New York, NY, USA, v. 11, n. 3, p. 309–346, 2002.

MOON, J. Y.; SPROULL, L. Essence of distributed work: The case of the linux kernel. **First Monday**, v. 5, 04 2000.

NAHM, U. Y.; MOONEY, R. J. Text mining with information extraction. In: STANFORD CA. **Proceedings of the AAAI 2002 Spring Symposium on Mining Answers from Texts and Knowledge Bases**. [S.l.], 2002. p. 60–67.

POMBO, N.; TEIXEIRA, R. Contribution of temporal sequence activities to predict bug fixing time. In: IEEE. **2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)**. [S.l.], 2020. p. 1–6.

ROBLES, G. et al. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In: **Proceedings of the 11th Working Conference on Mining Software Repositories**. [S.l.: s.n.], 2014. p. 222–231.

SORBO, A. D. et al. "won't we fix this issue?"qualitative characterization and automated identification of wontfix issues on github. **arXiv preprint arXiv:1904.02414**, 2019.

SRINIVASAN, K.; FISHER, D. Machine learning approaches to estimating software development effort. **IEEE Transactions on Software Engineering**, IEEE, v. 21, n. 2, p. 126–137, 1995.

SRIVASTAVA, A. N.; SAHAMI, M. **Text mining: Classification, clustering, and applications**. [S.l.]: CRC press, 2009.

TERCEIRO, A.; RIOS, L. R.; CHAVEZ, C. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In: IEEE. **2010 Brazilian Symposium on Software Engineering**. [S.l.], 2010. p. 21–29.

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

YU, Y. et al. Wait for it: Determinants of pull request evaluation latency on github. In: IEEE. **2015 IEEE/ACM 12th working conference on mining software repositories**. [S.l.], 2015. p. 367–371.

ZHANG, H.; GONG, L.; VERSTEEG, S. Predicting bug-fixing time: an empirical study of commercial software projects. In: IEEE PRESS. **Proceedings of the 2013 international conference on software engineering**. [S.l.], 2013. p. 1042–1051.

ZHANG, Y. et al. Exploring the use of@-mention to assist software development in github. In: **Proceedings of the 7th Asia-pacific symposium on internetware**. [S.l.: s.n.], 2015. p. 83–92.

ZHANG, Y. et al. ilinker: a novel approach for issue knowledge acquisition in github projects. **World Wide Web**, Springer, v. 23, n. 3, p. 1589–1619, 2020.