UNIVERSIDADE FEDERAL DO ACRE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

PREVISÃO DE INTEGRAÇÃO DE PULL REQUESTS EM PROJETOS OPEN-SOURCE

RIO BRANCO
2019

MATEUS DA SILVA COSTA

PREVISÃO DE INTEGRAÇÃO DE PULL REQUESTS EM PROJETOS OPEN-SOURCE

Monografia apresentada como exigência parcial para obtenção do grau de bacharel em Sistemas de Informação da Universidade Federal do Acre.

Profº. Orientador: Dr. Manoel Limeira de Lima Júnior Almeida.

RIO BRANCO

2019

TERMO DE APROVAÇÃO

MATEUS DA SILVA COSTA

PREVISÃO DE INTEGRAÇÃO DE PULL REQUESTS EM PROJETOS OPEN-SOURCE

Esta monografia foi apresentada como trabalho de conclusão de Curso de Bacharelado em Sistemas de Informação da Universidade Federal do Acre, sendo aprovado pela banca constituída pelo professor orientador e membros abaixo mencionados.

Compuseram a banca:

_____

Prof. Dr. Manoel Limeira de Lima Júnior Almeida.
Curso de Bacharelado em Sistemas de Informação

_____

Prof. Dr. Macilon Araújo Costa Neto.
Curso de Bacharelado em Sistemas de Informação

_____

Prof. Dr. Daricelio Moreira Soares.
Curso de Bacharelado em Sistemas de Informação

Rio Branco, 16 de dezembro de 2019.

A Gilvan Costa, Maria Ronise e Maria Eva.

AGRADECIMENTOS

Este trabalho marca a conclusão de uma grande etapa de minha vida, não sendo possível de ser completa sem a ajuda e subsídio de muitas pessoas. Agradeço sobretudo a minha família, a meus pais Maria Ronise e Gilvan, a minha vó Maria Eva e a todos os tios e tias que de muitas formas diferentes me ajudaram e apoiaram.

Agradeço aos meus amigos que me ajudaram nos momentos complicados e me apoiaram tecnicamente e psicologicamente.

A todos os professores e professoras que compartilharam desde os primeiros anos da escola seu conhecimento, deixo minha gratidão, principalmente ao professor Manoel Limeira por ter se disponibilizado a orientar este trabalho e por responder a todas as constates dúvidas.

“As vezes é preciso aprender a correr antes de começar a andar” - Tony Stark

RESUMO

O desenvolvimento distribuído de software *open-source* tem no modelo baseado em *pull requests* uma importante ferramenta para que desenvolvedores externos possam contribuir no projeto de software. Um *pull request* deve ser analisado por um integrador, membro da equipe principal do projeto, que decidirá pela sua integração (aceitação ou rejeição). A análise de *pull requests* pode ser demorada em decorrência da complexidade das mudanças, isso consome tempo do integrador que poderia ser utilizado para desenvolvimento. Por ser um processo decisório, pode ser útil realizar a previsão sobre a integração de um *pull request*. Este cenário já foi explorado em trabalhos anteriores, porém com uma abordagem diferente tanto para os atributos como para os algoritmos propostos. O objetivo principal deste trabalho se consiste em verificar se novos atributos melhoram a previsão de integração de *pull requests.* Para a realização dos experimento buscou-se seguir as fases do processo de descoberta de conhecimento em base de dados. Além disso, foi realizada a calibragem do método de avaliação, dos algoritmos e de estratégias de seleção de atributos. Essas atividades possibilitaram uma comparação dos conjuntos de dados obtidos com os utilizados em trabalhos anteriores. Como resultado, o conjunto de dados proposto por esse trabalho se mostrou ligeiramente melhor, com 83,51% de acurácia em média.

Palavras-chaves: *pull request*, mineração de dados, *open-source*, desenvolvimento distribuído, controle de mudanças, classificação.

ABSTRACT

Distributed development of open source software has the pull requests model as an important tool for external developers to contribute to software design. A pull request must be reviewed by an integrator, member of the core project team, who will decide on its integration (acceptance or rejection). Pull request analysis can be time consuming because of the complexity of the changes, this takes the integrator time that could be used for development. Because it is a decision-making process, it may be useful to forecast the integration of a pull request. This scenario has already been explored in previous works, but with a different approach to both attributes and proposed algorithms. The main objective of this work is to verify if new attributes improve the pull requests integration forecast. For the accomplishment of the experiments it was sought to follow the phases of the process of discovery of knowledge in database. In addition, the evaluation method, algorithms and attribute selection strategies were calibrated. These activities made it possible to compare the data sets obtained with those used in previous works. As a result, the data set proposed by this work was slightly better, with 83.51% accuracy on average.

Key-words: pull request, data mining, open source, distributed development, change control, classification.

LISTA DE FIGURAS

FIGURA 1 - METODOLOGIA DO TRABALHO.....18
FIGURA 2 - WORKFLOW REPOSITÓRIO COMPARTILHADO.....24
FIGURA 3 - PROCESSO DE UM PULL REQUEST.....27
FIGURA 4 - TELA DE DISCUSSÃO DE UM *PULL REQUEST* DO PROJETO BUNDLE NO GIT HUB.....28
FIGURA 5 - ETAPAS PRINCIPAIS DO PROCESSO DE KDD.....30
FIGURA 6 - DESLOCAMENTO DA JANELA DO MÉTODO TRAINING-TESTE SLIDING VALIDADION.....33
FIGURA 7 - DESLOCAMENTO DA JANELA DO MÉTODO TRAINING-TESTE SLIDING VALIDADION.....34
FIGURA 8 - DESLOCAMENTO ACUMULATIVO DA JANELA DO MÉTODO TRAINING-TEST SLIDING VALIDATION.....34
FIGURA 9 - MÉDIA DE ACURÁCIA DOS PROJETOS POR TAMANHO DA JANELA DE TREINO.....45
FIGURA 10 - RELEVÂNCIA DOS ATRIBUTOS PARA O ALGORITMO RANDOM FOREST.....51

LISTA DE QUADROS

QUADRO 1 - ESTRUTURA DE UMA MATRIZ DE CONFUSÃO.....35
QUADRO 2 - ATRIBUTOS DO CONJUNTO A REFERENTES AO *PULL REQUEST*.41
QUADRO 3 - ATRIBUTOS DO CONJUNTO A REFERENTES AO PROJETO.....42
QUADRO 4 - ATRIBUTOS DO CONJUNTO A REFERENTES AS RELAÇÕES SOCIAIS.....42
QUADRO 5 - ATRIBUTOS DO CONJUNTO B.....43

LISTA DE TABELAS

TABELA 1 - INFORMAÇÕES SOBRE OS PROJETOS PRESENTES NA BASE DE DADOS.....40
TABELA 2 - RESULTADOS PARA OS VALORES DE PARÂMETRO DO ALGORITMO IBK.....47
TABELA 3 - RESULTADOS PARA OS VALORES DE PARÂMETRO DO ALGORITMO J48.....47
TABELA 4 - RESULTADOS PARA OS VALORES DE PARÂMETRO DO ALGORITMO RANDOM FOREST.....47
TABELA 5 - RESULTADOS PARA OS VALORES DE PARÂMETRO DO ALGORITMO SMO.....48
TABELA 6 - RESULTADOS PARA OS VALORES DE PARÂMETRO DO ALGORITMO PART.....48
TABELA 7 - EXPERIMENTO COM A MELHOR CONFIGURAÇÃO DOS ALGORITMOS.....48
TABELA 8 - EXPERIMENTO COMPARATIVO ENTRE OS CONJUNTOS D1 E D2.51
TABELA 9 - EXPERIMENTO COMPARATIVO ENTRE OS CONJUNTOS A, B E D2 53

SUMÁRIO

1 INTRODUÇÃO.....13
1.1 PROBLEMA DA PESQUISA.....14
1.2 OBJETIVOS DA PESQUISA.....15
1.2.1 OBJETIVO GERAL.....15
1.2.2 OBJETIVOS ESPECÍFICOS.....16
1.3 JUSTIFICATIVA DA PESQUISA.....16
1.4 METODOLOGIA.....17
1.5 ORGANIZAÇÃO DO ESTUDO.....18
2 FUNDAMENTAÇÃO TEÓRICA.....19
2.1 GERÊNCIA DE CONFIGURAÇÃO.....19
2.1.1 SISTEMAS DE GERENCIAMENTO DE MUDANÇAS E CONTROLE DE VERSÃO.....21
2.1.2 FLUXOS DE TRABALHO EM SCVD.....23
2.1.3 *PULL REQUEST*.....25
2.2 *KNOWLEDGE DISCOVERY IN DATABASES* (KDD).....28
2.2.1 SELEÇÃO E PRÉ-PROCESSAMENTO.....30
2.2.2 TRANSFORMAÇÃO.....31
2.2.3 MINERAÇÃO DE DADOS.....31
2.2.4 ALGORITMOS DE CLASSIFICAÇÃO.....36
2.2.5 INTERPRETAÇÃO E AVALIAÇÃO.....37
2.2.6 FERRAMENTAS DE APOIO AO KDD.....37
3 ESTUDO DE CASO: PREVISÃO DE INTEGRAÇÃO DE *PULL REQUEST*.....39
3.1 SELEÇÃO, PRÉ-PROCESSAMENTO E TRANSFORMAÇÃO DOS DADOS 39
3.2 MINERAÇÃO DE DADOS E INTERPRETAÇÃO DOS RESULTADOS.....44
3.2.1 CALIBRAGEM DO MÉTODO *TRAINING-TEST SLIDING VALIDATION* 44
3.2.2 CALIBRAGEM DOS ALGORITMOS.....46
3.2.3 SELEÇÃO DE ATRIBUTOS.....50
3.2.4 COMPARAÇÃO ENTRE AS ABORDAGENS.....52
4 CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES.....55

4.1 CONSIDERAÇÕES FINAIS.....55
4.2 RECOMENDAÇÕES.....56
REFERÊNCIAS.....57

1 INTRODUÇÃO

Softwares sofrem mudanças constantes, o surgimento de *bugs* ou de novos requisitos dão origem a novas implementações, e consequentemente, a uma nova versão do software. Esse processo acontece diversas vezes de modo que um software pode ser definido como um conjunto de versões, onde o gerenciamento dessas versões e a maneira como elas são integradas é responsabilidade de uma área da engenharia de software denominada gerenciamento de configuração (SOMMERVILLE, 2011).

Na literatura existem diversas definições das atividades que compõem a gerência de configuração, tal como em Murta (2006), Pressman (1995) e Sommerville (2011), todavia ao menos duas atividades estão presentes em todas as definições a de controle de versão e controle de mudanças. O controle de versões busca gerenciar as distintas versões de objetos de configuração através da combinação de procedimentos e ferramentas, enquanto o controle de mudanças visa relacionar procedimentos humanos e ferramentas automatizadas para gerar mecanismos que possibilitem o controle das mudanças.

Algo comum a ambas as atividades é a utilização de ferramentas que automatizem os procedimentos. Nesse contexto, os *Version Control Systems* (Sistemas de Controle de Versão - SCV) e os *Issue Tracking System* (Sistema de Controle de Mudanças - SCM) são as ferramentas que dão suporte as atividades de gerenciamento de configuração e possibilitam um processo de desenvolvimento mais organizado (PRESSMAN, 1995; SOMMERVILLE, 2011).

No contexto do desenvolvimento distribuído de software, o controle de mudanças se

torna ainda mais complexo, o processo de controlar o acesso aos objetos de configuração e o controle das mudanças paralelas passam a lidar com um número maior de solicitações e conflitos, uma vez que o número de membros da equipe se torna maior e mais diversificado.

Os SCV's evoluiriam gradativamente para se adequar as necessidades do desenvolvimento distribuído de software e culminaram nos Sistemas de Controle de Versão Distribuídos (SCVD) (CHACON; STRAUB, 2014). Essas ferramentas unem características do controle de versão, tais como o versionamento dos objetos de configuração, com as do controle de mudanças, ou seja, o controle de acesso e a sincronização de mudanças.

As características dos SCVD's os tornaram uma das principais ferramentas de suporte ao processo de desenvolvimento distribuído, especialmente no contexto de projetos *open-source*, esse tipo de projeto possui o princípio de domínio público de modo que qualquer pessoa pode copiá-lo, editá-lo e distribui-lo (BAR; FOGEL, 2003).

Ao obter uma cópia do código fonte, um desenvolvedor externo pode fazer alterações sem que elas afetem o desenvolvimento do projeto principal, caso esse desenvolvedor queira se tornar um contribuidor enviando suas alterações ao projeto principal é necessário submeter uma solicitação que será analisada pelos membros da equipe principal do projeto, essa solicitação é denominada *pull request* (GOUSIOS; PINZGER; DEURSEN, 2014).

1.1 PROBLEMA DA PESQUISA

O modelo de desenvolvimento de projetos *open-source* baseado em *pull requests* se tornou extremamente popular e utilizado em uma grande quantidade de projetos (BARR et al., 2012). Todavia, para que a integração de *pull request* ocorra, é necessário que um dos membros da equipe faça a avaliação das alterações do código para averiguar qual será o impacto no projeto, apenas após isso será possível tomar uma decisão.

Segundo Lima Júnior (2017), no contexto *open-source,* o tempo entre a solicitação de integração e a decisão de aceitá-la ou rejeitá-la pode variar muito, indo de alguns minutos

até 30 dias. Segundo Gousios, Pinzer e Deursen (2014), em 2013, no GitHub[1], projetos como Ruby, On Rails e Homebrew receberam mais de cinco mil *pull requests.* Essa enorme quantidade influencia o ritmo de trabalho dos desenvolvedores que são responsáveis pela análise de integração (integradores) (YU et al., 2015).

Os integradores devem tomar decisões rápidas e corretas para tornar o processo de *pull request* o mais eficiente possível seja reduzindo o tempo do processo ou buscando garantir a qualidade do código aceito (GOUSIOS et al., 2015). Por ser um processo de tomada de decisão, realizar a previsão da aceitação pode ser extremamente útil para torná-lo mais eficiente.

Trabalhos anteriores já exploraram as características do *pull request* em Gousios, Pinzer e Deursen (2014) obteve-se uma taxa de 86% de acurácia em projetos *open-source* no GitHub utilizando-se de algoritmos de classificação. Em trabalhos relacionados a mineração de dados, variações dos atributos que compõem as bases de dados e as abordagem de avaliação dos algoritmos podem influenciam melhoras nos resultados. Sendo assim é possível que com uma nova abordagem desse tipo, as taxas de acertos obtidos em Gousios, Pinzer e Deursen (2014) possam ser aumentadas. Deste modo questiona-se: É possível melhorar a taxa de acerto na previsão de integração de um *pull request*?

1.2 OBJETIVOS DA PESQUISA

Esta subseção aborda o objetivo geral do trabalho, bem como os objetivos específicos.

1.2.1 OBJETIVO GERAL

O objetivo deste trabalho é verificar se novos atributos melhoram a previsão de integração de *pull requests.*

[1]https://github.com

1.2.2 OBJETIVOS ESPECÍFICOS

Visando alcançar o objetivo geral deste trabalho foram definidos os seguintes objetivos específicos:

a) Selecionar projetos e atributos que iram compor a base de dados;

b) Preprocessar a base de dados;

c) Calibrar a configuração dos algoritmos de classificação e o método de avaliação para o contexto da pesquisa;

d) Selecionar as melhores configurações para a previsão de integração de *pull request*;

e) Avaliar os algoritmos de classificação;

f) Analisar os resultados obtidos.

1.3 JUSTIFICATIVA DA PESQUISA

De acordo com Bar e Fogel (2003), há 80% de chance de se está utilizando total ou parcialmente um software livre quando se envia um e-mail, ou 65% quando se navega por páginas na web. A importância dos softwares *open-source* aumenta a cada dia com a sua utilização direta ou indireta assim como com o interesse de contribuir com esses projetos.

A natureza compartilhada de projetos *open-source,* a popularidade desse modelo de desenvolvimento e o grande número de projetos que o adotam faz com que vários contribuidores ajudem no desenvolvimento desses projetos, principalmente de forma distribuída, tornando alta a quantidade de *pull requests* recebidos em projetos de código aberto (GOUSIOS; PINZGER; DEURSEN, 2014).

O VSCode (Visual Studio Code) é um editor de texto de código aberto da Microsoft, seu código fonte está disponível em um repositório no GitHub. Entre 28 de agosto a 4 de setembro de 2019, ou seja, em 7 dias, esse projeto recebeu 46 *pull requests* que precisam ser

analisados, para receber um estado final: aceito ou rejeitado. Essa análise pode ser rápida ou demorada, dependendo da complexidade das alterações realizadas o que aumenta o tempo de desenvolvimento do projeto.

Prever a integração de um *pull request* pode contribuir para a tomada de decisão, o que pode agilizar o processo. Em Gousios, Pinzer e Deursen (2014), se utiliza atributos sobre *pull requests* no GitHub, como objetivo de identificar fatores que contribuem para a integração de *pull request*.

Deste modo, este trabalho se justifica pela possibilidade de melhorar os resultados já existentes através da análise de novos atributos, de modo que os resultados obtidos possam ajudar integradores de *pull request* na análise, tornando o processo mais rápido e eficiente.

1.4 METODOLOGIA

Segundo Wazlawick (2009), o desenvolvimento de pesquisas na área da computação pode ser confuso de classificar ou delimitar uma metodologia em detrimento de ser uma área nova da ciência e em constante mudança. O autor define cinco categorias de trabalhos: Apresentação de um produto, apresentação de algo diferente, apresentação de algo presumivelmente melhor, apresentação de algo reconhecidamente melhor e apresentação de um prova.

Na apresentação de algo diferente a forma de se resolver um problema se difere das demais categorias, busca-se realizar uma comparação de técnicas com uma apresentação mais objetiva dos resultados (WAZLAWICK, 2009). Este trabalho se classifica como uma apresentação de algo diferente, uma vez que outros trabalhos já abordaram a previsão de integração de *pull requests*, todavia em outro contexto e com métodos e dados diferentes.

Para Gil (1991), para se classificar uma pesquisa em geral é importante utilizar-se de algum critério, o autor estabelece dois principais: os objetivos gerais e os procedimentos técnicos utilizados.

Quanto aos seus objetivos esse trabalho pode ser classificado como uma pesquisa exploratória, uma vez que se busca alcançar maior compreensão a respeito do processo de

integração de um *pull request.* Embora a previsão da integração já tenha sido explorada por outros autores, este trabalho busca aprimorando o conhecimento já existente explotando diferentes técnicas para melhorar a taxa de acerto na previsão de integração de *pull requests*.

No que se refere ao procedimento técnico adotado este trabalho pode ser classificado como uma pesquisa *ex-post-facto* por ser baseada na realização de um experimento sobre um caso de uso (GIL, 1991). Cinco epatas metodológicas foram definidas para esse trabalho e são ilustradas na Figura 1.

Figura 1 - Metodologia do trabalho

Fonte: Elaboração própria.

As etapas metodológicas seguem o fluxo do processo de descoberta de conhecimento em base de dados (*Knowledge Discovery in Databases*– KDD) esse processo define cinco etapas para a extração de conhecimento a partir de um conjunto de dados, o processo de KDD e suas etapas são tratados na seção 2.2.

1.5 ORGANIZAÇÃO DO ESTUDO

Este trabalho possui além deste mais três capítulos. O capítulo 2 aborda os dois conceitos que fundamental essa pesquisa: gerência de configuração na seção 2.1 e o processo de KDD na seção 2.2.

O terceiro capítulo traz em seu conteúdo os resultados obtidos, a primeira seção deste capítulo trata das etapas de seleção, pré-processamento e transformação dos dados, enquanto a segunda seção apresenta as etapas de mineração de dados e a interpretação dos resultados obtidos.

No quarto e último capítulo são realizadas as considerações finais e as recomendações para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está divido em duas seções que apresentam os principais conceitos para o entendimento do trabalho. A seção 2.1 trata dos conceitos da gerência de configuração abordando na subseção 2.1.1 os dois principais sistemas desta área: o sistema de gerenciamento de mudanças e o sistema de controle de versão. Na subseção 2.1.2, são apresentados os dois fluxos de trabalhos utilizados nos Sistemas de Controle de Versão Distribuídos (SCVD). A subseção 2.1.3, detalha o conceito de *pull request*.

A seção 2.2 trata do processo *Knowledge Discovery in Databases (*KDD) utilizado como processo base desta pesquisa, a sua primeira subseção trata da etapa seleção e pré-processamento do KDD. A seção 2.2.2 aborda a etapa de transformação dos dados. A etapa de mineração de dados e suas características constituem o conteúdo da subseção 2.2.3. Os algoritmos utilizados pela tarefa preditiva de classificação são abordados na subseção 2.2.5. Na subseção 2.2.6, é apresentado um resumo da etapa de interpretação e avaliação do KDD. A subseção 2.2.7 apresenta as ferramentas que apoiam o KDD.

2.1 GERÊNCIA DE CONFIGURAÇÃO

Um software é um produto demasiadamente instável e extremamente suscetível a

mudanças. Ao longo do processo de desenvolvimento ou utilização de um produto de software pode ocorrer o surgimento de *bugs,* alterações nos requisitos ou até a necessidade de adequação a novos hardwares. Todas essas mudanças podem ser convertidas em novas implementações no software gerando uma nova versão (SOMMERVILLE, 2011).

As inevitáveis mudanças aumentam o nível de confusão entre os engenheiros de software que trabalham em um projeto, para Pressman (1995) a não realização de análise, registros, relatos e controle sobre as mudanças são o que originam essas confusões.

A gerência de configuração surge como uma forma de gerenciar, controlar e registrar essas mudanças ao longo de todo o processo de engenharia do software. De acordo com Sommerville (2011), esta área está associada com politicas, processos e ferramentas que asseguram esse controle, uma vez que é fácil perdê-lo, o que impossibilita a identificação de quais mudanças e versões de componentes foram incorporadas.

O processo de gerência de configuração deve ser implantado para garantir que os artefatos mantenham a qualidade, a consistência e a atualidade ao longo da vida dos projetos, os artefatos que devem ser controlados incluem a documentação, os modelos, e o código fonte (FILHO, 2012).

De acordo com Filho (2012), para cumprir seu objetivo a gerência de configuração faz uso dos seguintes meios:

a) Identificação dos itens de configuração;

b) Controle de alterações e das versões desses itens;

c) Manutenção da integridade e rastreabilidade deses itens.

Esses objetivos podem ser alcançados através de um conjunto de tarefas estabelecidas pelo processo de gerenciamento de configuração de software que tem por objetivo principal controlar as mudanças. Pressman (1995), define cinco tarefas que devem estar presentes no processo de gerência de configuração, são elas:

a) **Identificação de objetos na configuração de software:** Visa facilitar a organização identificando item de configuração que devem ser nomeados separadamente através de um abordagem orientada a objetos, de modo que cada objeto tenha características que os identifiquem;

b) **Controle de versão:** Essa tarefa tem por objetivo gerenciar as diferentes versões

dos objetos de configuração através da combinação de procedimentos e ferramentas;

c) **Controle de mudanças:** Busca combinar procedimentos humanos e ferramentas automatizadas para controlar as mudanças, através de dois principais elementos: o controle de acesso e o controle de sincronização;

d) **Auditoria de configuração:** Visa complementar uma revisão técnica formal, no âmbito da avaliação de um objeto de configuração e de suas características que geralmente não são consideradas durante a revisão técnica formal;

e) **Relato de status:** Tem por objetivo a construção de um fluxo de informações sobre cada nova alteração, de modo a responder questões como: O que aconteceu? Quem o fez? Quando aconteceu? O que mais será afetado?

Na literatura existem outras divisões das tarefas que compõem o processo de gerência de configuração. Sommerville (2011) define quatro atividades que estariam presentes nesse processo: (i) Gerenciamento de mudança; (ii) Gerenciamento de versões, (iii) Construção do sistema; e (iv) Gerenciamento de *releases*. Para Murta (2006), esse processo pode ser dividido em três sistemas principais: (i) Controle de modificações; (ii) Controle de versões; e (iii) Gerenciamento de construção.

Partindo da visão dos autores é possível afirmar que esse processo envolve ao menos duas atividades principais: o gerenciamento de mudanças e controle de versões. Essas atividades são amparadas por ferramentas, uma vez que o controle pode se tornar demasiadamente complexo para um humano realizar.

2.1.1 SISTEMAS DE GERENCIAMENTO DE MUDANÇAS E CONTROLE DE VERSÃO

Como já mencionado anteriormente, os softwares estão submetidos as inevitáveis mudanças, seja em decorrência de *bugs* ou por necessidades evolutivas. Essas mudanças não podem ocorrer de forma desordenada, para garantir que sua aplicação ao sistema ocorra de maneira controlada se torna necessário a definição de um processo de gerenciamento de

mudanças, amparado por ferramentas (SOMMERVILLE, 2011).

Não só os desenvolvedores de um software podem identificar necessidades de mudanças, no caso de um software em operação qualquer *stakerholder* pode identificar um *bug*, ou uma nova funcionalidade que precise ser implementada. Para Pressman (1995), o controle de mudança se inicia com um pedido de mudanças que passa por uma avaliação técnica que analisa os impactos e efeitos colaterais sobre os demais itens de configuração. O resultado da avaliação deve compor um relatório de mudanças que é utilizado como embasamento por uma autoridade controladora de mudanças para a criação de uma ordem de mudança de engenharia, que descreve a mudança a ser feita, as restrições e os critérios para revisão e auditoria.

Sommerville (2011), afirma que um *Change Request* (CR), ou solicitação de mudança, pode descrever sintomas de um *bug* ou um pedido para a adição de uma nova funcionalidade. Os *stakerholders* devem submeter suas solicitações através de um formulário, o C*hange Request Form* (CRF) que formalizam o processo. Os CRF funcionam como uma carteira de identidade para a mudança, acompanhando e registrando cada decisão tomada ao longo do processo.

Além de registrar o requerimento de mudanças o CRF armazena informações relativas a estimativas de custos, recomendações e datas de quando o CR foi realizado, aprovado, implementado e validado.

Com a implementação desse processo em sistemas computacionais surgem os SCM que armazenam todas as informações necessitarias para a identificação dos objetos de configuração envolvidos, quais mudanças foram realizadas, por quem e em qual contexto (MURTA, 2006). Os SCM implementam cada uma das etapas do processo de gerência de mudanças e são utilizados em muitos projetos onde os usuários realizam os CR através das chamadas *issues*. Como exemplos de SCM tem-se: Bugzilla[2], Mantis[3], Redmine[4] e Trac[5].

No contexto de controle de versão, existem os Sistemas de Controle de Versão (SCV), que dão suporte ao gerenciamento de versões automatizando e armazenando atributos que caracterizam uma nova versão (PRESSMAN, 1995; SOMMERVILLE, 2011).

[2]https://www.bugzilla.org/
[3]https://www.mantisbt.org/
[4]https://www.redmine.org/
[5]https://trac.edgewall.org/

Inicialmente os SCV consistiam em simples bancos de dados locais que armazenavam todas as informações sobre as alterações. Todavia, a necessidade de se desenvolver de maneira colaborativa levou ao desenvolvimento dos SCV Centralizados, onde o versionamento ocorre em um único servidor acessado por todos os desenvolvedores (CHACON; STRAUB, 2014).

Os SCV Centralizados são amplamente utilizados, porém possuem a grande desvantagem de ter as informações armazenadas em um único lugar, de modo que caso haja a perda do servidor todas as informações são perdidas. Segundo Chacon e Straub (2014), é nesse contexto que atuam os Sistemas de Controle de Versão Distribuídos (SCVD), em sistemas como esses os clientes não apenas podem vê a situação dos arquivos, mas também podem criar uma cópia completa desse repositório em sua área de trabalho local.

Existem muitos SCVD's disponíveis, tais como Git[6], Mercurial[7], Bazaar[8] e Darcs[9]. O Git é um dos mais utilizados por servir como base para o site GitHub, um serviço de controle de versão e mudanças online que hospeda muitos projetos privados e principalmente os de código aberto.

Segundo Chacon e Straub (2014) o GitHub é um serviço amplamente utilizado por milhares de projetos *open-source* e constitui-se como o maior *host* único para repositórios Git, possibilitando, dentre outras coisas, o rastreamento de problemas e a revisão de código. Apesar de não ser uma parte direta do projeto Git, o GitHub é o principal meio de interação entre desenvolvedores e essa ferramenta (CHACON; STRAUB, 2014). Esse serviço consegue unir características dos SCM com os SCVD de modo a ser uma ferramenta robusta para o gerenciamento de configurações.

2.1.2 FLUXOS DE TRABALHO EM SCVD

Segundo Bird e Zimmermann (2012), o advento SCVD facilita o processo de

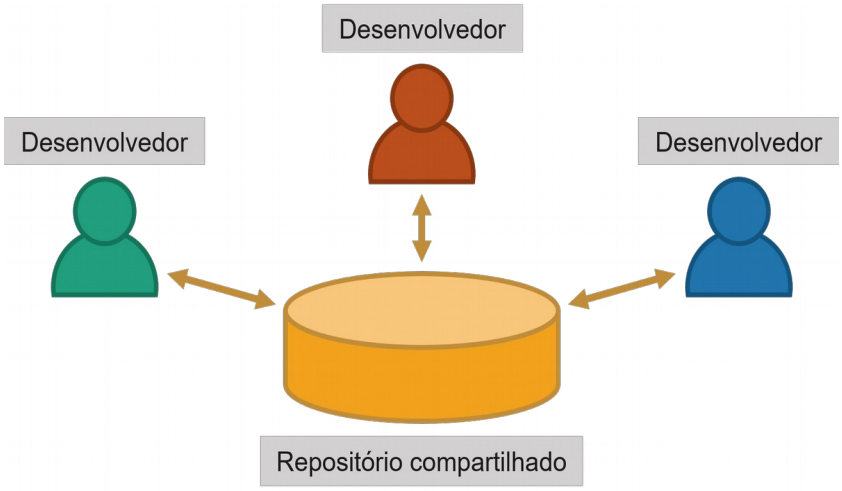
[6]https://git-scm.com/
[7]https://www.mercurial-scm.org/
[8]https://bazaar.canonical.com/
[9]http://darcs.net/

branching (ramificação) e fusão de versões de um projeto que esteja sendo desenvolvidas simultaneamente. Deste modo muitos projetos, principalmente os *open-source,* se utilizam desses sistemas para gerenciar o desenvolvimento e evolução do código fonte e demais produtos de software gerados. Em projetos utilizam essa ferramenta, de acordo com Chacon, Straub (2014) e Lima Júnior (2017), existem dois principais fluxos de trabalho ou *workflows*: o centralizado e o de gerente de integração.

No *workflow* centralizado, os membros da equipe principal de desenvolvimento realizam a operação de *clone,* onde copiam o repositório do projeto inteiro para suas máquinas e realizam as alterações nesse repositório local através da operação de *commit,* operação que registra um conjunto de alterações em seu repositório local. Uma vez que fazem parte da equipe principal do projeto eles podem realizar a operação *push*, operação responsável por enviar as alterações locais para o repositório do projeto (CHACON; STRAUB, 2014).

A Figura 2 exemplifica a forma como os diversos desenvolvedores interagem com um repositório compartilhado, a utilização de ferramentas de gerência de configuração auxiliam no registro das mudanças causadas pelas operações e todas as informações relevantes para um possível auditoria ou recuperação para um estado anterior.

Figura 2 - Workflow repositório compartilhado



Fonte: Adaptado de Chacon e Straub (2014).

Segundo Lima Júnior (2017), no *workflow* gerente de integração o repositório do projeto encontra-se disponível para que qualquer desenvolvedor, mesmo os que não fazem parte do projeto possam copiar o seu conteúdo através da operação de *fork*, onde uma instância do repositório do projeto é criada, porém agora com permissão de escrita para o desenvolvedor externo.

Após a realização de uma operação de *fork,* o desenvolvedor pode realizar as operações de *clone*, *commit* e *push* de modo semelhante ao repositório compartilhado. Todavia, as modificações só ficaram efetivas na instância do repositório principal pertencente ao desenvolvedor, ou seja, seu repositório local, para que elas sejam enviadas ao projeto é necessário a realização de um *pull request*, uma solicitação de integração das mudanças realizadas (CHACON; STRAUB, 2014; LIMA JÚNIOR, 2017).

Para que o processo de integração seja realizado, um membro da equipe principal deve analisar as alterações e determinar se elas devem ou não ser aceitas e incorporadas ao repositório principal. Nesse caso, a integração através de um *merge*, caso o parecer seja negativo, um *feedback* pode ser enviado ao solicitante ou o *pull request* pode ser rejeitado (*closed*), esse processo é tratado com maiores detalhes na subseção 2.2 deste capítulo.

Segundo Chacon e Straub (2014), as facilidades propiciadas pelas operações descritas, bem como a possibilidade de um desenvolvedor trabalhar nas alterações do software em seu repositório local até que seja realizado um *merge* por parte do integrador, faz com que o fluxo de trabalho gerente de configuração seja muito utilizado. Plataformas como BitBucket[10], GitHub, Google Code[11] implementam essas funcionalidades e são muito utilizadas por projetos *open-source* na distribuição de código fonte e no desenvolvimento colaborativo.

2.1.3 *PULL REQUEST*

Com a popularização da filosofia *open-source*, os adeptos desta começaram a se deparar com problema logístico da engenharia de software, não era suficiente para um desenvolvedor de um software desse tipo apenas disponibilizá-lo em um local da internet. Somente a disponibilização impossibilitaria aos usuários a possibilidade de contribuir com a evolução do software, seja pela identificação de *bugs* ou por mudanças no próprio código fonte (BAR; FOGEL, 2003).

No desenvolvimento privado onde o código fonte não é livre, as empresas possuem

[10]https://bitbucket.org/
[11]https://code.google.com/

um time conhecido de desenvolvedores e o trabalho e dividido de maneira organizada, de modo que cada um possa identificar o trabalho como um todo. Todavia, nos projetos *open-source*, não há restrições de quem tem acesso ou quanto as mudanças que serão realizadas, para um desenvolvedor ou para uma equipe de desenvolvimento pequena gerenciar essas mudanças e desenvolver o código se torna um desafio (BAR; FOGEL, 2003).

O paradigma de desenvolvimento distribuído, propiciado pelos SCVD e em especial pelo fluxo de trabalho gerente de integração, possibilita aos desenvolvedores colaborar em projetos *open-source* de maneira mais eficiente (GOUSIOS; PINZGER; DEURSEN, 2014). Esse processo de colaboração exige uma constante interação entre os desenvolvedores externos que querem contribuir e a equipe do projeto, isso ocorre devido ao fato de que os colaboradores não tem permissão de escrita no repositório principal (CHACON; STRAUB, 2014).

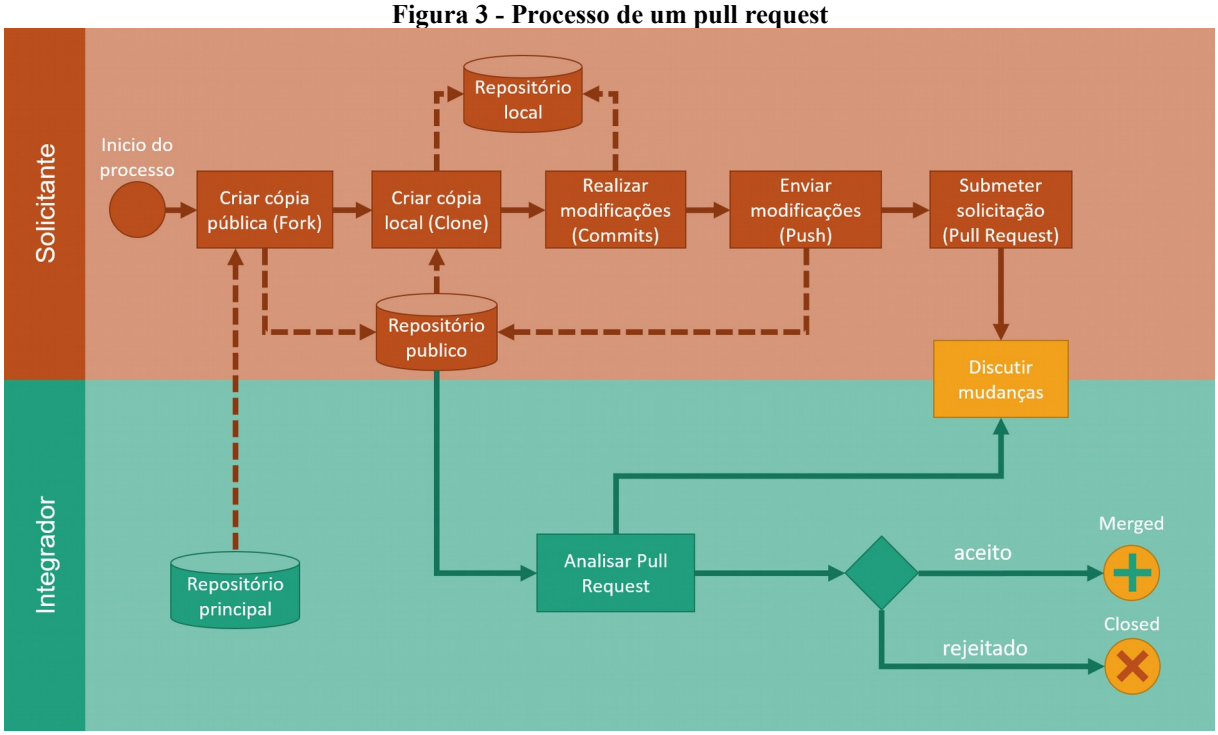
Nesse contexto, para que um desenvolvedor externo possa contribuir com o projeto é necessário o envio de uma solicitação de integração das mudanças, realizadas nos diferentes artefatos, ao repositório principal do projeto, essa solicitação recebe o nome de *pull request* (LIMA JÚNIOR, 2017; SOARES, 2017).

De acordo com Gousios, Pinzger e Deursen (2014), no GitHub, 14% dos projetos hospedados fazem uso desse modelo de contribuição. Os *pull requests* podem ser utilizados tanto por desenvolvedores externos como por membros da equipe principal, que apesar de poderem escrever diretamente no repositório principal, podem sistematizar uma avaliação das mudanças realizas pela própria equipe (LIMA JÚNIOR, 2017).

Um *pull request* pode ser definido como um conjunto de operações de *commit*, cada um contem um conjunto de mudanças em um ou mais artefatos do projeto que podem envolver correção de *bugs*, refatoração de código ou novas funcionalidades (GOUSIOS; PINZGER; DEURSEN, 2014; MADDILA; BANSAL; NAGAPPAN, 2019).

O processo de integração de um *pull request* esta inserido dentro do fluxo de trabalho gerente de integração e seu processo está ilustrado na Figura 3. Segundo Soares (2017), após criar uma cópia do repositório principal com uma operação de *fork*, é possível criar um repositório local (*clone*) onde modificações podem ser realizadas com *commits*. Após a realização das alterações, o contribuidor (Solicitante) abre (*open*) um *pull request*, avisando a equipe do projeto das suas alterações possibilitando assim que um dos membros da equipe principal (Integrador) possa avaliar o trabalho realizado.

Figura 3 - Processo de um pull request

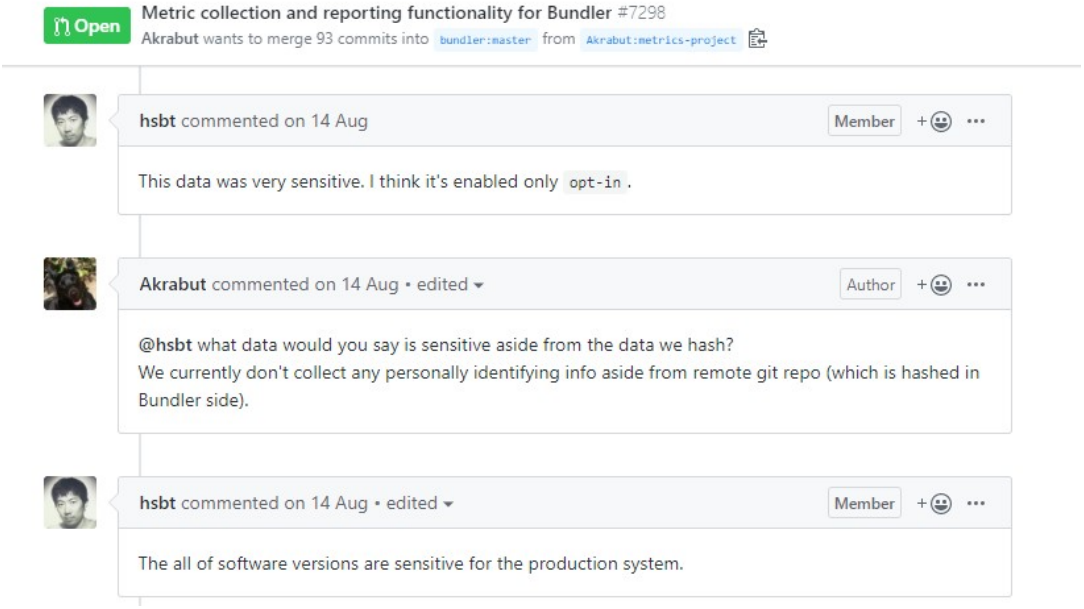


Fonte: Adaptado de Lima Júnior (2017) e Soares (2017).

Para Lima Júnior (2017) um *pull request* pode adotar três possíveis estados: *open*, quando a solicitação é enviada e uma decisão ainda não foi tomada; *merged*, caso as alterações são aceitas para integração ao repositório; ou *closed*, quando a integração é rejeitada.

Plataformas como o GitHub propiciam uma forma dos envolvidos se comunicarem a respeito das alterações, principalmente quando há algo de errado nos artefatos alterados, de modo que o contribuidor possa realizar novos *commits* para resolver. A Figura 4 apresenta uma tela de discussão sobre um *pull request*, nela é possível visualizar que o usuário, e membro da equipe do projeto, “hsbt” discute com o autor “Akrabbut” sobre as alterações propostas.

Figura 4 - Tela de discussão de um *pull request* do projeto Bundle no Git Hub



Fonte: Elaboração própria.

O processo descrito pode envolver vários desenvolvedores e alterações, de modo que uma grande quantidade de atributos são armazenados. Soares (2017) afirma que dados como título, descrição, identificação do contribuidor, número de *commits*, número de arquivos e linhas de código modificadas são uma importante fonte de informações sobre o contexto do desenvolvimento colaborativo de software.

Deste modo um *pull request* pode ser visto não só como um facilitador do processo de desenvolvimento, mas também como um importante instrumento de análise, para a identificação de padrões que possam contribuir para a melhora do processo de desenvolvimento em diversos projetos.

2.2 *KNOWLEDGE DISCOVERY IN DATABASES* (KDD)

A capacidade de armazenamento e processamento aumentou drasticamente desde o final do século XXI, essa grande disponibilidade de recursos computacionais faz com que haja cada vez mais sistemas operando, em vários lugares e nos mais diversos contextos, o que gera uma quantidade gigantesca de dados (HAN; KAMBER; PEI, 2012).

Transações de vendas, manutenção de estoque, produção industrial, perfis de

consumidores ou fornecedores, entre outras informações são geradas diariamente em escalas que um ser humano não conseguiria analisar ou registrar. Nesse contexto, a possibilidade de se extrair conhecimento desses dados fomentou uma série de esforços que vão desde novas teorias computacionais até novas ferramentas, dentre esses esforços se destaca o processo de KDD (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996a).

Fayyad, Piatetsky-Shapiro e Smyth (1996) explicam que o modelo tradicional de extração de conhecimento de uma base de dados consistem na análise manual por parte de um especialista. Todavia, com a era da informação, a grande quantidade de dados disponíveis torna inviável para um ser humano analisá-los sem o auxílio computacional. Assim, o processo de KDD surge como uma forma de solucionar essa questão.

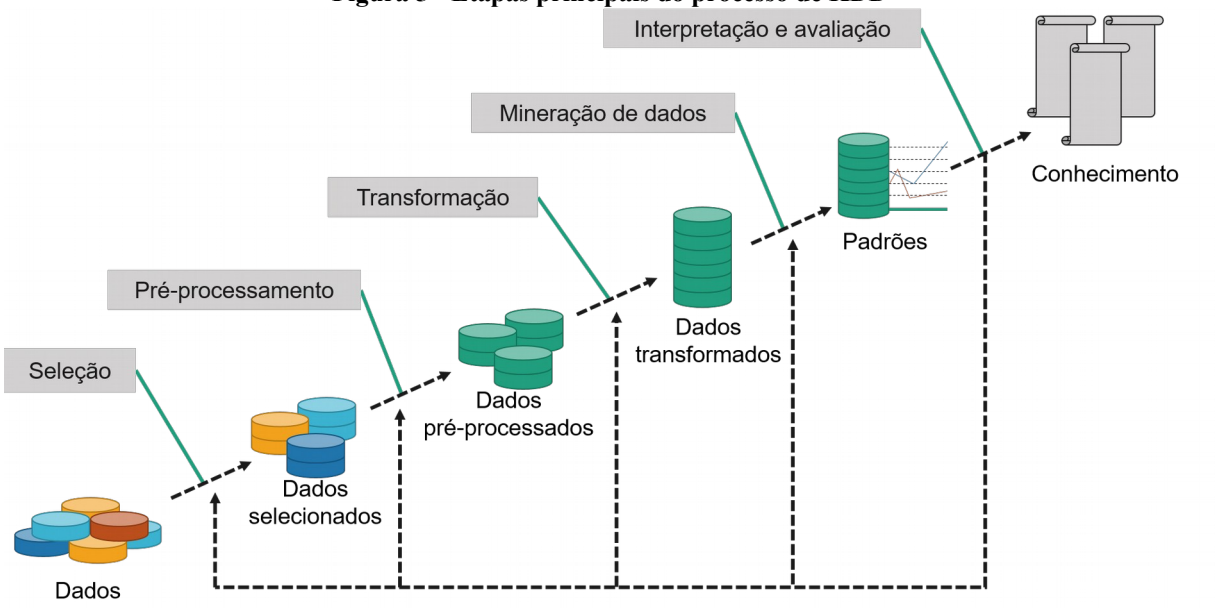
A mineração de dados pode ser descrita como a busca por padrões em um conjunto de dados (WITTEN; FRANK; HALL, 2011). A ligação entre a definição do KDD e o de mineração é explorada de diversas maneiras na literatura. Para Han, Kamber e Pei (2012) e Zhou (2003) as essas definições são sinônimos, já par Cios et al. (2007) e Fayyad, Piatetsky-shapiro e Smyth (1996a) o KDD é um processo composto por etapas e a mineração de dados é uma destas.

De acordo com Camilo (2009), uma definição muito utilizada na literatura é a de Fayyad, Piatetsy-Sapiro e Smyth (1996a), segundo essa definição, o KDD é um processo não trivial de identificação de padrões novos, validos, potencialmente úteis e compreensíveis em uma base de dados, essa definição será tomada como base para esse trabalho. Nesse contexto, uma base de dados consiste em um conjunto de fatos ou registros, e os padrões são alguns modelos nos quais os dados podem ser encaixados ou a identificação de uma estrutura.

Como um processo, o KDD possui uma estrutura formada por um conjunto de etapas que devem ser seguidas por quem executa um projeto de descoberta de conhecimento. O modelo deve definir os procedimentos que devem ser seguidos em cada etapa, os modelos de KDD podem ser classificados entre os de pesquisa acadêmica e os industriais (CIOS et al., 2007).

Fayyad, Piatetsy-Sapiro e Smyth (1996a) apresenta um modelo básico composto por cinco etapas principais: Seleção, pré-processamento, transformação, mineração de dados e Interpretação e avaliação. Essas etapas constituem um processo incremental e são ilustradas na Figura Figura 5. Destaca-se na representação o caráter iterativo do processo, possibilitando o retorno a uma das etapas anteriores caso seja necessário.

Figura 5 - Etapas principais do processo de KDD



Fonte: Adaptado de Fayyad, Piatetsy-Sapiro e Smyth (1996).

De acordo com Cios et al. (2007) esse modelo de processo é classificado com0o de pesquisa acadêmica e é considerado um dos principais desse tipo, com base nisso esse modelo foi escolhido para esta pesquisa. As etapas do processo de KDD são tradadas nas subseções seguintes.

2.2.1 SELEÇÃO E PRÉ-PROCESSAMENTO

O processo de KDD tem início a partir da seleção dos dados que irão ser analisados, para tal, é necessário compreender o domínio do que está sendo analisado, identificando quais são as metas do processo (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996a).

Após a compreensão do contexto dos dados, deve-se seguir para a escolha de quais dados iram compor a base de dados, para Dantas e Pessoa (2008) essa massa de dados pode ser um conjunto ou subconjunto de variáveis. Essa fase não está apenas ligada a seleção dos dados, mas também com a sua extração a partir de um banco de dados (TRONCHONI et al., 2010).

Com os dados selecionados a próxima etapa é o pré-processamento, nesta etapa os dados são “limpos”, para tal se faz uso de processos que visam reduzir dados que possam ser

irrelevantes ou que atrapalhem as etapas seguintes.

O objetivo dessa “limpeza” é assegurar a qualidade dos dados, através de operações como remover ruídos ou *outliers.* Também é possível realizar a definição das estratégias para modelar e contabilizar os dados, assim como lidar com valores ausentes. Outras operações também envolvem a retirada ou resolução conflitos entre os tipos de dados presentes na base (DANTAS; PESSOA, 2008; FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996b).

2.2.2 TRANSFORMAÇÃO

Esta etapa precede a etapa de Mineração de Dados, e busca estabelecer o padrão de dados que serão utilizados na mineração. Para que a transformação seja realizada deve-se primeiro realizar a seleção dos registros e atributos relevantes (GOLDSCHMIDT; PASSOS, 2005). Em uma base de dados sobre moda por exemplo, em que se deseja identificar o interesse atual do mercado, dados referentes a 2000 possivelmente não serão relevantes e podem enviesar os resultados para padrões não uteis.

Após selecionar os dados que precisam ser transformados, para que seja possível a aplicação dos algoritmos, a transformação se utiliza da redução das dimensões para diminuir o número efetivo de variáveis em consideração ou para encontrar representações invariantes para os dados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996b).

Para Rodrigues (2018) essa etapa é importante devido à existência de algoritmos de mineração que não funcionam com alguns tipos de dados e a possibilidade de alguns algoritmos oferecerem melhores resultados com a presença de certos tipos de dados.

2.2.3 MINERAÇÃO DE DADOS

Segundo Witten, Frank e Hall (2011), a mineração de dados pode ser definida como a busca por padrões em dados. Esse termo é tratado por muitos como sinônimo do processo de

KDD, porém a mineração de dados pode ser considerada como parte do processo de KDD. Nessa etapa, busca-se a identificação de padrões válidos e que apresentem alguma utilidade compreensível (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996a; HAN; KAMBER; PEI, 2012).

Para a execução dessa etapa, inicialmente deve-se escolher qual tarefa ou técnica de mineração será executada, comumente essas tarefas são divididas entre dois tipos: descritivas e preditivas (WITTEN; FRANK; HALL, 2011).

As tarefas descritivas também chamadas de não supervisionadas focam na busca por padrões de relacionamento entre os dados. Esse tipo de tarefa não necessita de um atributo alvo, logo não há a necessidade de realizar categorização. Ao invés disso, a estratégia utilizada nesse tipo de mineração envolve a tentativa de medir a similaridade entre os dados, exemplos comumente utilizados são a extração de regras de associação e agrupamento (CIOS et al., 2007; HAN; KAMBER; PEI, 2012).

As tarefas preditivas buscam prever o valor de um determinado atributo alvo. Esse tipo de tarefas se utiliza dos valores de outros atributos para realizar a previsão, se tornando necessário que eles estejam associados ao atributo alvo. Os algoritmos de previsão podem ainda ser subdivididos entre os de classificação, quando o atributo alvo e discreto ou nominal, e regressão, quando o atributo alvo é numérico (CIOS et al., 2007; HAN; KAMBER; PEI, 2012; HAND; MANNILA; SMYTH, 2001; LIMA JÚNIOR, 2017).

A tarefa preditiva de classificação tem como entrada um conjunto de dados, e como saída um modelo de classificação. Essa tarefa constitui-se como um processo sistemático que envolve duas etapas: treino e teste. O treino envolve a construção de um modelo a partir dos dados rotulados, ou seja, a característica ou classe que busca-se prever está presente na base de dados de modo a possibilitar a descoberta do padrão que a prevê. Na próxima etapa o modelo é testado sem que os rótulos das classes estejam conhecidos (HAN; KAMBER; PEI, 2012; LIMA JÚNIOR, 2017).

Os algoritmos de regressão também fazem uso da etapa de treino e teste, todavia, a tarefa de regressão busca determinar uma função que relacione o conjunto de variáveis independentes e a variável dependente (o alvo) com o objetivo de determiná-la como um valor numérico (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996b; LIMA JÚNIOR, 2017).

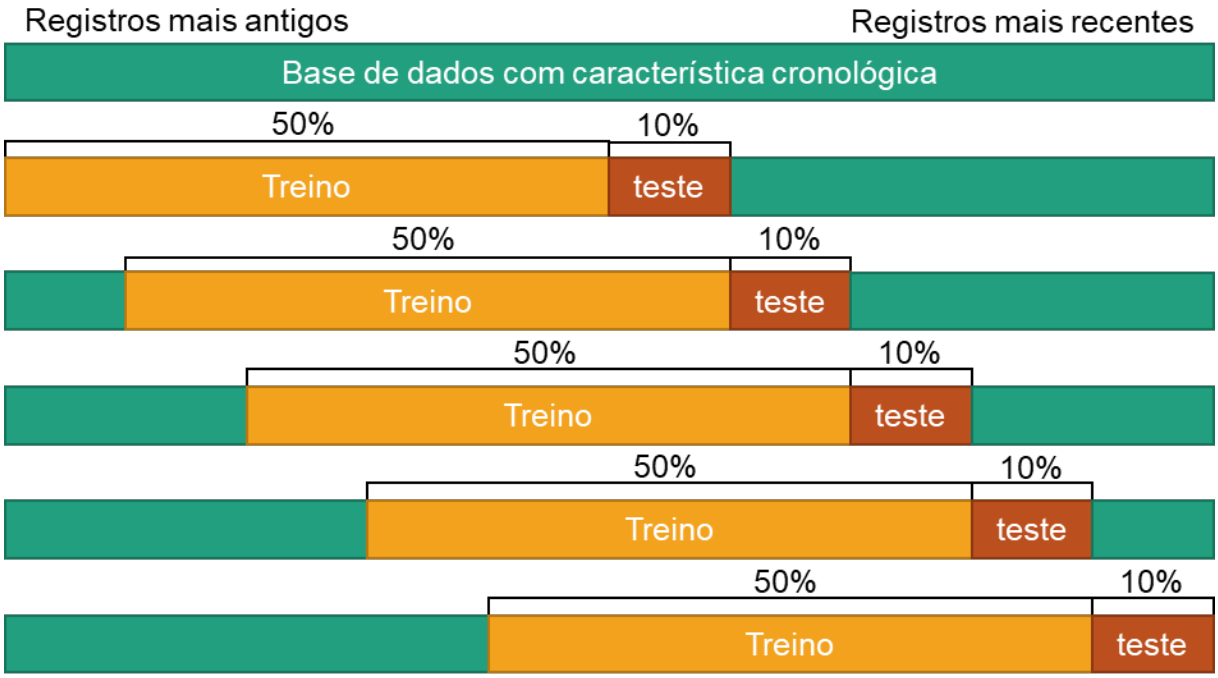
Assim é possível concluir que a tarefa classificação é mais apropriada para prever o resultado da integração de um *pull request*, uma vez que o resultado só pode assumir um valor discreto (*Closed* ou *Merged*), os principais algoritmos desta tarefa são tratados na subseção 2.2.4.

A definição de como será dividido os conjuntos de treino e teste são determinados por métodos de amostragem, que variam esses valores afetando a acurácia (*accuracy)*, a taxa de acerto dos modelos, possibilitando uma avaliação diferente dos algoritmos.

Um dos principais métodos é o *holdout* que consiste em dividir a base de dados aleatoriamente em duas, tipicamente dois terços para treino e um terço para teste. Uma derivação desse método é o *random subsampling*, esse método consiste na execução do *holdout* várias vezes, o valor de acurácia geral é dado pela média. Outro método comum é o *croos-validation*, nela a base de dados é separada aleatoriamente em *k* partições, com aproximadamente o mesmo tamanho, a partir dai o treino e teste são executados *k* vezes sendo que cada partição é ao menos uma vez utilizada para teste e as restantes para treino (HAN; KAMBER; PEI, 2012).

Em situações em que a ordem dos dados é importante, os métodos de amostragem descritos podem não ser adequado, como no caso de dados ordenados cronologicamente em que registros do futuro podem ser utilizados para prever dados do passado. Para esse contexto, Lima Júnior (2017) propõe o método *training-test sliding validation* que mantêm os dados ordenados cronologicamente, particionando a base em várias “janelas”, para que então ocorra a divisão entre treino e teste. A Figura Figura 7 representa o deslocamento da janela para uma base de dados com atributos cronológicos, a janela ocupa um total de 60% da base, uma vez que 50% são destinados a treino e 10% para teste. Uma característica desse método é o deslocamento da janela partindo dos registros mais antigos para os mais recentes, esse deslocamento é igual ao tamanho de teste o que faz com que os registros anteriormente utilizados para teste também sejam utilizados para treino (LIMA JÚNIOR, 2017; LIMA, 2017).

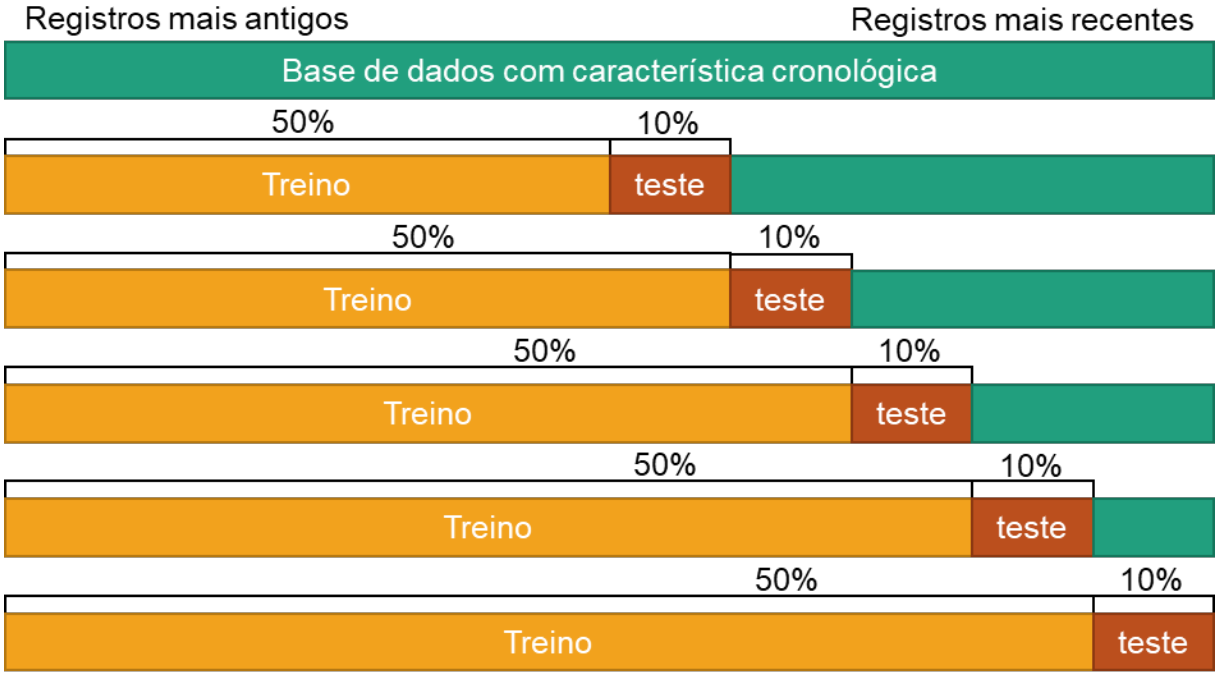
Figura 7 - Deslocamento da janela do método training-teste sliding validadion



Fonte: Adaptado de Lima (2017).

Uma possibilidade do *training-test sliding validation* é acumular os registros de treino, conforme ilustra a Figura 8, o tamanho de treino representado em amarelo engloba as instâncias mais antigas a medida que o deslocamento da janela ocorre (LIMA, 2017).

Figura 8 - Deslocamento acumulativo da janela do método training-test sliding validation



Fonte: Adaptado de Lima (2017).

Após a definição do método que será utilizado e da execução da mineração de dados com um algoritmo classificação é possível visualizar o desempenho através da Matriz de

Confusão, que apresenta o número de previsões corretamente ou incorretamente classificadas. O Quadro 1 apresenta a estrutura que uma Matriz de Confusão, para um problema com classificação binaria, deve seguir. De acordo com Han, Kamber e Pei (2012) os resultados podem ser agrupados entre os que foram corretamente classificados (Verdadeiro Positivo e Verdadeiro Negativo) e os incorretamente classificados (Falso Positivo e Falso Negativo).

Quadro 1 - Estrutura de uma Matriz de Confusão

Classe Verdadeira	**Classe Predita**	
	Verdadeira	**Negativa**
Verdadeira	Verdadeiro Positivo (VP)	Falso Negativo (FN)
Negativa	Falso Positivo (FP)	Verdadeiro negativo (VN)

Fonte: Adaptado de Han, Kamber e Pei (2012).

A partir da matriz de confusão é possível determinar o número de elementos positivos (P) através da Equação 1 e negativos (N) a partir da Equação 2.

$$P = TP + FN \tag{1}$$

$$N = FP + VN \tag{2}$$

Para Han, Kamber e Pei (2001), o desempenho do classificador pode ser aferido através de algumas métricas. A principal delas é a acurácia, que representa a porcentagem de elementos do conjunto de teste que foram classificados como sua classe verdadeira, esse valor pode ser obtido através da Equação 3. O *Recall* ou valor de Cobertura consiste na proporção de elementos positivos classificados como positivos, a Equação 4 é utilizada para obtê-lo. A Equação 5 é utilizada para se obter o valor de precisão (*Precision*) que é a proporção de elementos classificados como positivos e que realmente são positivos. Outra métrica de desempenho que é possível de se obter é a média harmônica entre a *Precision* e o *Recall* (Equação 6).

$$\frac{TP + TN}{P + N} \tag{3}$$

$$\frac{TP}{P} \tag{4}$$

$$\frac{TP}{TP + FP} \tag{5}$$

$$\frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

2.2.4 ALGORITMOS DE CLASSIFICAÇÃO

Na tarefa de classificação busca-se prever o valor de um atributo alvo, também chamada de classe, isso pode dar-se de diversas maneiras e adotando-se várias estratégias, de modo que há diversos algoritmos de classificação, de diferentes paradigmas, que podem ser bons para um domínio e ruim para outros. Alguns dos algoritmos mais comuns são:

a) **Naive Bayes (NB):** É um algoritmo classificador baseado no Teorema de Bayes, esse algoritmo calcula a probabilidade de cada uma das instâncias pertencer a cada uma das classes (LIMA JÚNIOR, 2017).

b) **IBk (Instance Based Learner):** Este classificador é uma variação do algoritmo *k-Nearest Neighbors* (k-NN) que calcula à distância de cada nova instância referente as instâncias já existentes, classifica-se o novo registro com a classe mais comum dos registros com menor distância (GOLDSCHMIDT; PASSOS, 2005).

c) **J48:** Este algoritmo se utiliza de árvore de decisão, sendo que para cada um dos nós não folha existe uma condição do tipo “se <condição> então <conclusão>” relacionando um atributo e um grupo de valor (GOLDSCHMIDT; PASSOS, 2005).

d) **Random Forest:** É um método de classificação que combina um grupo de vários modelos de classificação onde cada um dos grupos é uma árvore de decisão. Para realizar a classificação é feito uma votação majoritária entre as classes referentes a cada árvore (HAN; KAMBER; PEI, 2012).

e) **PART:** É um algoritmo para indução de regras que combina os algoritmos C4.5 e RIPPER, porém ao contrário destes não há a execução de operações globais para produzir um conjunto preciso de regras, sendo a simplicidade sua maior vantagem. Esse algoritmo adota a estratégia de “dividir para conquistar”, removendo instâncias anteriores e criando regras para as restantes até que não haja mais instâncias (FRANK; WITTEN, 1998).

f) **SMO (Sequential Minimal Optimization):** É um algoritmo baseado em *Support Vector Machine* (SVM), ele decompõe um problema quadrático em

subproblemas quadráticos. Em cada passo da execução o algoritmo escolhe o menor problema de otimização possível através da escolha de dois multiplicadores de Lagrange (PLATT, 1998).

2.2.5 INTERPRETAÇÃO E AVALIAÇÃO

Uma vez que um algoritmo foi aplicado a Mineração de Dados resulta na obtenção de padrões. Todavia, padrões não implicam em um conhecimento direto, para a consolidação desses padrões é necessário interpretá-los e avaliar os resultados obtidos.

Para Fayyad, Piatetsy-Sapiro e Smyth (1996b) uma revisão criteriosa deve ser realizada, removendo padrões redundantes ou irrelevantes. A avaliação e interpretação dos resultados obtidos devem ser analisados de forma crítica, especialista de dados, conhecedores do domínio e tomadores de decisão devem participar dessa etapa, aqui o uso de ferramentas de prospecção gráfica são essências para facilitar a visualização dos modelos obtidos (CAMILO, 2009).

A interpretação é influenciada pela compreensão do domínio dos dados, prever a desvalorização do Real referente ao Dólar pode ser um incentivo a compra de produtos brasileiros por empresas de outros países, porém pode ser um sinal de crise econômica para o governo.

Caso os resultados obtidos não sejam uteis ou satisfatórios é possível retornar a qual quer etapa anterior, ou mesmo realizar todo o processo novamente graças ao fator iterativo desse processo (RODRIGUES, 2018).

2.2.6 FERRAMENTAS DE APOIO AO KDD

Várias ferramentas facilitam a execução das etapas de KDD, apresentando facilidades como o controle dos dados por interface gráfica e um conjunto de algoritmos já

implementados. Alguns exemplos de ferramentas que permitem essas facilidades são: Orange[12], RStudio[13], RapidMiner[14], Anaconda[15] e Weka[16] (RODRIGUES, 2018).

A ferramenta Orange reúne um conjunto de ferramentas que facilitam a visualização de dados e a execução de experimentos de mineração de dados e *machine learning*. RStudio é um ambiente que facilita a utilização da linguagem R é voltado para a criação de gráficos e execução de cálculos relacionados a estatística. RapidMiner possui em sua plataforma o suporte ao pré-processamento e a várias técnicas de *machine learning*. Anaconda gerencia pacotes que possibilitam a instalação de pacotes das linguagens Python e R voltados para a área de ciência de dados. Weka é uma das mais populares ferramentas de descoberta de conhecimento, possuindo uma vasta coleção de algoritmos voltados para tarefas descritivas e preditivas de mineração de dados.

As ferramentas envolvidas nas etapas dessa pesquisa foram o Weka e o RStudio. A ferramenta Weka foi escolhida devido as facilidades do pré-processamento, a grande variedade de algoritmos disponíveis (WITTEN; FRANK; HALL, 2011). Outra vantagem da Weka é a existência de uma extensão que implementa o método de escolha de amostragem *training-test sliding validation,* tratado na subseção 2.2.4, proposto por Lima Júnior (2017) e implementada por Lima (2017) e que facilita a execução da mineração de dados em bases de dados com atributos cronológicos.

[12]http://orange.biolab.si/
[13]https://rstudio.com/
[14]https://rapidminer.com/
[15]https://anaconda.org/
[16]https://www.cs.waikato.ac.nz/ml/weka/

3 ESTUDO DE CASO: PREVISÃO DE INTEGRAÇÃO DE *PULL REQUEST*

Este capítulo apresenta os passos seguidos nesta pesquisa, bem como os resultados obtidos. O KDD foi utilizado como processo base sendo que as fases foram aglutinadas para melhor compreensão do que foi realizado. Na seção 3.1 deste capítulo são detalhadas as atividades realizadas nas fases de seleção e pré-processamento dos dados e atributos que compuseram a base dados, assim como a descrição dos dados transformados. Na seção 3.2, os resultados das atividades executas na fase de mineração de dados são mostrados e é realizado a interpretação dos mesmos.

3.1 SELEÇÃO, PRÉ-PROCESSAMENTO E TRANSFORMAÇÃO DOS DADOS

Os dados utilizados nesta pesquisa foram obtidas mediante extração realizada com a ferramenta GHTTorrent por Lima Júnior (2017) e utilizada para realizar a previsão de tempo de vida e integrador mais apropriado para um *pull request*, a base de dados também foi utilizada em Lima Júnior et al. (2019) também na previsão de tempo de vida de *pull request.* A base de dados possui informações sobre *pull requests* e relações sociais entre os desenvolvedores envolvidos de 30 projetos hospedados no Git Hub.

Uma seleção inicial de atributos foi aplicada para a retirada de informações e

projetos que não contribuíssem para o contexto da pesquisa ou que de alguma forma não tivessem informações válidas.

A base de dados selecionada possui 83 atributos sobre 30 projetos, todavia alguns projetos não poderiam ser utilizados nessa pesquisa, sendo que 10 dos projetos foram descartados por questões relativas a falta de atributos ou por estarem com caracteres especiais que impossibilitaram a execução com a ferramenta Weka. Deste modo, a pesquisa se utilizou dos 20 projetos descritos na Tabela 1, os projetos selecionados são de cinco linguagens de programação diferentes e totalizam 48.883 *pull requests*.

Tabela 1 - Informações sobre os projetos presentes na base de dados

Linguagens	Projetos	Número de *pull requests*	Aceitos (%)	Rejeitados (%)
Java	candlepin	1093	87,65	12,35
	hazelcast	1833	91,27	8,73
	netty	2028	49,70	50,30
	okhttp	1194	88,27	11,73
JavaScript	appium	1984	87,85	12,15
	meteor	1299	57,51	42,49
	node	2851	33,36	66,64
	pouchdb	1610	75,34	24,66
Python	kuma	3565	89,20	10,80
	pulp	2403	94,13	5,87
	rosdistro	10155	93,76	6,24
	scikit learn	3213	31,93	68,07
Ruby	bundler	1327	77,32	22,68
	diaspora	2612	76,65	23,35
	metasploit framework	5342	86,20	13,80
	nancy	1293	84,38	15,92
	vagrant	1622	77,00	23,00
Scala	marathon	1360	87,06	12,94
	scala ide	1050	86,95	13,05
	scala js	1049	94,18	5,82
	Média	2444,15	77,49	22,51

Fonte: Elaboração própria.

Em média 77,49% dos *pull requests* tiveram sua integração aceita, enquanto 22,51% foram rejeitados. A maioria dos projetos possuem mais *pull requests* aceitos do que rejeitados,

com exceção dos projetos scikit-learn e node, os vinte projetos constituem o conjunto A.

O pré-processamento dos atributos foi realizado para retirada daqueles sem importância ou que apresentassem informações que não estivessem presentes no momento do envio do *pull request*. Foram retirados atributos como identificadores, métricas de arquivos abertos depois do envio do *pull request* e tempo de vida. O conjunto A possui ao todo 52 atributos, o Quadro 5 apresenta nome e descrição de 24 atributos relativos ao *pull request.*

Quadro 2 - Atributos do conjunto A referentes ao *pull request*

Id	Nome	Descrição
A1	created_at_week_day	Dia da semana em que o *pull request* foi enviado
A2	created_at_day_turn	Turno do dia em que o *pull request* foi enviado
A3	conflict	Se a palavra “conflito” aparece nos comentários do *pull request*
A4	forward_links	*Se o pull request* tem links para outros *pull requests*
A5	intra_branch	Se o *pull request* está entre as ramificações do mesmo repositório
A6	description_length	Número de caracteres na descrição do *pull request*
A7	num_commits	Número de commits no *pull request*
A8	files_added	Quantidade de arquivos adicionados pelo *pull request*
A9	files_deleted	Quantidade de arquivos removidos pelo *pull request*
A10	files_modified	Quantidade de arquivos modificados pelo *pull request*
A11	files_changed	Quantidade de arquivos alterados pelo *pull request*
A12	src_files	Arquivos de código fonte alterado pelo *pull request*
A13	doc_files	Arquivos de documentação alterados pelo *pull request*
A14	other_files	Outros arquivos alterados pelo *pull request*
A15	src_churn	Número de linhas alteradas pelo *pull request*
A16	test_churn	Número de linhas de teste alteradas
A17	new_entropy	Taxa de desordem inserida pelo *pull request*
A18	entropy_diff	Taxa de desordem inserida por arquivo
A19	commits_on_files_touched	Porcentagem de commits por arquivo afetado pelo *pull request*
A20	commits_to_hottest_file	Número de *commits* para os arquivos de teste mais alterados
A21	hotness	Média de *commits* nos arquivos envolvidos pelo *pull request* em relação a todos os *commits* do projeto nos últimos 3 meses
A22	at_mentions_description	Se há uma menção a um usuário na descrição do *pull request*
A23	at_mentions_comments	Se há uma menção a um usuário nos comentários do *pull request*

Fonte: Elaboração própria.

No conjunto A, 11 atributos são referentes a informações do projeto para qual o *pull request* foi enviado e são descritos no Quadro 3.

Quadro 3 - Atributos do conjunto A referentes ao projeto

Id	Nome	Descrição
A24	prev_pull_reqs_project	Número de *pull requests* anteriores ao *pull request* submetido
A25	project_succ_rate	Taxa de aceitação de *pull requests* no projeto
A26	perc_external_contribs	Percentual de contribuições externas no projeto
A27	sloc	Número de linhas de código
A28	test_lines_per_kloc	Número de linhas de teste executáveis a cada 1000 linhas de código
A29	test_cases_per_kloc	Número de linhas de teste a cada 1000 linhas de código
A30	asserts_per_kloc	Número de *assert statements* a cada 1000 linhas de código
A31	stars	Número de estrelas do projeto
A32	team_size	Número de membros da equipe de desenvolvimento do projeto
A33	project_age	Tempo de vida do projeto
A34	workload	Número de *pull request* ainda abertos até o recebimento do *pull request* atual
A35	ci	Se o projeto usa integração contínua

Fonte: Elaboração própria.

Além de informações referentes ao *pull request* e projeto, também foram selecionados para o conjunto A, atributos referentes as relações sociais e do solicitante, esses atributos são descritos no Quadro 4.

Quadro 4 - Atributos do conjunto A referentes as relações sociais

Id	Nome	Descrição
A36	requester	Nome de usuário do solicitante
A37	prev_pullreqs	Número de *pull requests* submetidos antes do *pull request* atual
A38	requester_succ_rate	Taxa de sucesso do solicitante em *pull requests* anteriores
A39	followers	Número de usuários seguidos pelo solicitante
A40	following	Número de usuários que seguem o solicitante
A41	requester_age	Idade do solicitante
A42	main_team_member	Se o solicitante é membro do time principal do projeto
A43	watcher_project	Se o solicitante é um observador do projeto
A44	req_follows_integrator	Se o solicitante segue o integrador
A45	integrator_follows_req	Se o integrador segue o solicitante
A46	prior_interaction_issue_events	Número de vezes em que o solicitante interagiu com um *issue* do projeto antes da submissão do *pull request*
A47	prior_interaction_issue_comments	Número de comentários em *issue* do projeto antes da submissão do *pull request*

A48	prior_interaction_pr_events	Número de iterações em pull requests antes da submissão do *pull request*
A49	prior_interaction_pr_comments	Número de iterações em comentários do *pull request* antes da submissão do *pull request*
A50	prior_interaction_commits	Número de iterações em commits do *pull request* antes da submissão do *pull request*
A51	prior_interaction_commit_comments	Número de iterações em comentários de commits do *pull request* antes da submissão do *pull request*
A52	first_response	Tempo em minutos desde a criação do *pull request* até a primeira interação de um desenvolvedor

Fonte: Elaboração própria.

Foi criado ainda o conjunto B com os mesmos vinte projetos, porém com os mesmos atributos utilizados em Gousios, Pinzger e Deursen (2014), os atributos que compõem esse conjunto são listados no Quadro 5.

Quadro 5 - Atributos do conjunto B

Categoria	Id	Nome	Descrição
Pull request	B1	num_commits	Número de commits no *pull request*
	B2	src_churn	Número de linhas alteradas pelo *pull request*
	B3	test_churn	Número de linas de teste alteradas
	B4	files_changed	Número de arquivos afetados pelo *pull request*
	B5	num_comments	Discussões e comentários de revisão de código
	B6	num_participants	Número de participantes da discussão do *pull request*
	B7	conflict	A palavra conflito aparece nos comentários do *pull request*
	B8	forward_link	*Pull requests* tem links para outros *pull requests*
Projeto	B9	sloc	Linhas executáveis de código por tempo de *merge* do *pull request*
	B10	team_size	Número de membros ativos do projetos nos últimos 3 meses antes da criação do *pull request*
	B11	perc_ext_contribs	Proporção de *commits* de membros externos sobre a equipe principal nos últimos 3 meses
	B12	commits_files_touched	Número total de *commits* em arquivos afetados pelo *pull reques*, 3 meses antes de sua criação
	B14	test_lines_per_kloc	Um proxy para a cobertura de teste do projeto
Solicitante	B15	prev_pullreqs	Número de *pull requests* submetidos antes do *pull request* atual
	B16	requester_succ_rate	Taxa de *pull requests* do solicitante que foram aceitos

Fonte: Elaboração própria.

Quanto aos atributos transformados, o atributo created_at, que existia na base e referenciava a data e hora de envio do *pull request*, foi utilizado para criar os atributos A1 e

A2 do conjunto A. A classe foi criada a partir da transformação do atributo merged_at que armazenava a data e hora em que o *pull request* foi integrado ao projeto, de modo que sua ausência significa a rejeição no processo de integração.

Para fins de calibragen, foi criado o conjunto C constituído por cinco projetos: candlepin, appium, kuma, bundler e marathon. Cada um dos projetos pertencem a uma linguagem de programação diferente presente na base e possuem os mesmos atributos do conjunto A.

3.2 MINERAÇÃO DE DADOS E INTERPRETAÇÃO DOS RESULTADOS

A fase de mineração de dados proposta pelo processo de KDD, neste trabalho, foi realizada junto com a fase de interpretação e avaliação dos resultados. Essa decisão foi tomada devido ao fato de que a cada experimento uma avaliação seria imediatamente seguida da interpretação dos resultados. Isso também pode ser justificado pela quantidade de experimentos realizados nesta fase.

Com o objetivo de ser imparcial e justo durante a realização dos experimentos, realizou-se a calibragem do tamanho da janela de treino do método *training-test sliding validation* a partir da variação do tamanho da janela de treino, dos algoritmos utilizados a partir da variação do principal parâmetro de cada um e experimentos comparativos com diferentes conjuntos de dados. Como fim dessa fase, foi realizado um experimento comparativo com os diferentes conjuntos de atributos.

3.2.1 CALIBRAGEM DO MÉTODO *TRAINING-TEST SLIDING VALIDATION*

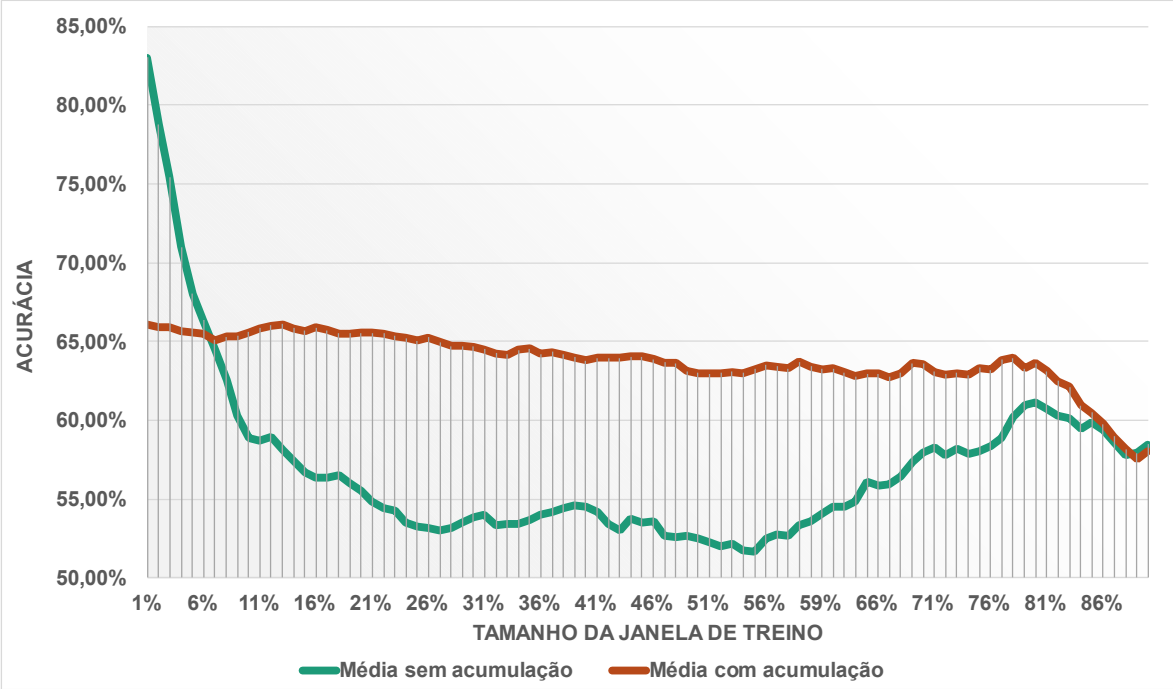
Para evitar qualquer viés nos resultados da pesquisa, realizou-se antes da avaliação dos algoritmos de classificação, a calibragem do método *training-test sliding validation.* Através da variação do tamanho da janela de treino. Em seguida, a calibragem dos algoritmos

foi realizada através da varição de seus principais parâmetros de configuração.

Como explicado na subseção 2.2.3, o *training-test sliding validation* é um método de avaliação para tarefas preditivas baseado na divisão de janelas de treino e teste e que possibilita preservar a característica temporal dos dados nos experimentos. Esse método pode utilizar a janela de treino acumulativa ou não, o quê também foi levado em consideração durante a sua calibragem. A calibragem consistiu na variação do tamanho da janela de treino, sendo a janela de teste fixada em 1% da base. A varição de treino iniciou-se com 1%, indo até 90% com o incremento de 1% a cada novo experimento, a calibragem foi executada com o conjunto C. O algoritmo utilizado neste experimento foi o Naive Bayes (NB), um dos algoritmos mais tradicionais para a tarefa de classificação.

A Figura 9 apresenta o gráfico dos resultados da calibragem do método *training-test sliding validation*. No gráfico é possível perceber que a variação com acumulação apresenta uma média do valor de acurácia superior no intervalo de tamanho de janela de treino entre os valores 7 e 89.

Figura 9 - Média de acurácia dos projetos por tamanho da janela de treino



Fonte: Elaboração própria.

Em decorrência da maior estabilidade nos valores de acurácia da variação com acumulação do método, optou-se por utilizar esta opção nos demais experimentos, sendo o valor de treino fixado em 13%, referente ao maior valor de acurácia obtido (66,10%) com 1% da base para teste.

3.2.2 CALIBRAGEM DOS ALGORITMOS

Após determinar o tamanho da janela de treino, buscou-se a calibragem dos algoritmos através da variação dos parâmetros de entrada. Foram selecionados 5 algoritmos de classificação para serem utilizados nesta pesquisa, são eles: IBk (*Instance Base Learner*), J48, Random Forest, SMO (*Sequential Minimal Optimization*) e PART. Esses algoritmos foram escolhidos de modo a pertencerem a diferentes tipos de paradigmas de aprendizagem: semelhança entre as instâncias, árvores de decisão, combinação de classificadores (ensemble), regras de associação e máquinas de vetores de suporte.

O algoritmo IBk possui como principal parâmetro a varição do atributo *k* que define a quantidade de vizinhos semelhantes e devem ser considerados para definir a classe ao qual a instância pertence, a variação acorreu com valores inteiros ímpares de 1 até 35. Para o algoritmo J48, o parâmetro testado foi o número mínimo de objetos que auxilia na criação da árvore de decisão, essa variação iniciou-se com 5 e foi incrementada em 5 até 60. No Random Forest, o parâmetro avaliado foi o número de iterações que se refere ao número de árvores que será utilizado na classificação, a variação teve incremento de 50 iterações, iniciando de 50 até 500. Para o algoritmo SMO foi realizada a variação com três *kernels*: PolyKernel, Puk e RBKKernel. No algoritmo PART, a variação utilizada foi o número mínimo de objetos assim como no J48, se utilizando de números pares partindo de 2 até 26.

Os experimentos com as variações de parâmetros ocorreram sobre o conjunto C. Para cada algoritmo foi criado um *ranking* que representa a posição da acurácia em relação aos outros algoritmos, ou seja, quanto menor a posição melhor a avaliação do algoritmo. O valor do *ranking* para cada um dos projetos foi utilizado para avaliar o desempenho dos algoritmos por meio da média dos *rankings.* Em caso de empate nos valores das acurácias, a média da soma das posições é utilizada como posição no *ranking.*

Foram utilizadas duas métricas para determinar o melhor valor de parâmetro: A média das acurácias e a média do *ranking*. O maior valor de média de acurácia e o menor valor de média do *ranking* para um valor de parâmetro caracterizam o melhor resultado e estão destacados em negrito.

A Tabela 2 mostra os valores das métricas para cada um dos valores do parâmetro *k* para o algoritmo IBk, por questões de visualização foram omitidos da tabela os resultados

para os valores de parâmetro entre 5 á 17 e 12 á 33. Observando as duas métricas é possível perceber que os valores de parâmetro 33 e 35 apresentam os mesmos valores de métricas e que esses são os melhores, por serem parâmetros igualmente bons considerou-se a menor quantidade de vizinho (33) como o melhor valor de parâmetro para o algoritmo IBk.

Tabela 2 - Resultados para os valores de parâmetro do algoritmo IBk

	Variação do k									
Métricas	**1**	**3**	**5**	**...**	**17**	**19**	**12**	**...**	**33**	**35**
Média da acurácia	83,97%	87,63%	88,53%	...	89,27%	89,31%	89,31%	...	**89,79%**	89,79%
Média do ranking	18	17	166	...	10,9	10	8,4	...	**2,3**	2,3

Fonte: Elaboração própria.

Os valores das métricas para cada valor de parâmetro do algoritmo J48 são apresentados na Tabela 3, é possível perceber que a maior média de acurácia e a menor média do *ranking* foram obtidas com o valor de parâmetro 50. Novamente, para melhor visualização não foram inseridos na tabela os resultados dos valores de parâmetro 20, 25, 40 e 45.

Tabela 3 - Resultados para os valores de parâmetro do algoritmo J48

	Variação do Número Mínimo de Objetos									
Métricas	**5**	**10**	**15**	**...**	**17**	**19**	**...**	**50**	**55**	**60**
Média da acurácia	84,72%	84,72%	84,56%	...	84,46%	84,78%	...	**85,50%**	85,24%	85,49%
Média do ranking	7,9	8,3	8	...	7,9	6,2	...	**3,9**	6,1	4,2

Fonte: Elaboração própria.

Na Tabela 4 são apresentados os resultados para cada valor de parâmetro do algoritmo Random Forest, é possível perceber na tabela que a maior média de acurácia é igual a 86,46% e a menor média do *ranking* é igual a 3,1, essas métricas indicam que o melhor valor de parâmetro é o 400.

Tabela 4 - Resultados para os valores de parâmetro do algoritmo Random Forest

	Variação do Número de Iterações								
Métricas	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Média da acurácia	86,33%	86,34%	86,28%	86,35%	86,42%	86,39%	**86,46%**	86,44%	86,42%
Média do ranking	5,6	5,5	7,4	5,6	3,9	4,8	**3,1**	4,4	4,7

Fonte: Elaboração própria.

Os resultados obtidos com a variação dos três *kernels* do algoritmo SMO são apresentados na Tabela 5. O *kernel* RBF foi considerado o melhor para o contexto desta

pesquisa, uma vez que apresentou as melhores métricas, sendo 87,02% a média de acurácia e 1,2 a média do *ranking.*

Tabela 5 - Resultados para os valores de parâmetro do algoritmo SMO

	Variação do *Kernel*		
Métricas	**PolyKernel**	**Puk**	**RBFKernal**
Média da acurácia	85,58%	86,97%	**87,02%**
Média do ranking	3	1,8	**1,2**

Fonte: Elaboração própria.

Para a variação de parâmetro do algoritmo PART foram obtidos os valores de métricas apresentado na Tabela 6. O valor 20 consolidou-se como o melhor valor de parâmetro por apresentar a maior média de acurácia e a menor média do ranking para esse algoritmo.

Tabela 6 - Resultados para os valores de parâmetro do algoritmo PART

	Variação do Número Mínimo de Objetos								
Métricas	**2**	**4**	**6**	**...**	**18**	**20**	**22**	**24**	**26**
Média da acurácia	88,13%	88,20%	88,05%	...	84,90%	**85,00%**	84,53%	84,53%	84,46%
Média do ranking	12,6	10,6	9,3	...	5,8	**4,3**	5,5	4,8	6,8

Fonte: Elaboração própria.

. O objetivo deste experimento foi identificar o melhor parâmetro para cada algoritmo no contexto da pesquisa. Para fins de comparação, os melhores resultados são apresentados na Tabela 7, também é possível observar as três métricas utilizadas para determinar o melhor algoritmo: Média da acurácia, média do *ranking* e o número de vitórias obtidas. Nota-se que há uma diversificação entre os valores de acurácia entre os projetos, por exemplo o projeto Meteor apresenta valores de acurácia abaixo de 65% para todos os algoritmos, já o projeto Scala JS apresenta valores de acurácia acima de 94% para todos os algoritmos. Isso demonstra que cada projeto possui peculiaridades e a importância de uma análise que as leve em consideração.

Tabela 7 - Experimento com a melhor configuração dos algoritmos

		Acurácia				
Linguagem	**Projeto**	**Ibk**	**J48**	**Random Forest**	**SMO**	**PART**
Java	candlepin	91,05%	84,00%	91,05%	91,05%	85,58%
	hazelcast	92,30%	92,30%	92,36%	92,30%	91,29%
	netty	69,26%	67,84%	71,76%	65,17%	68,07%
	okhttp	87,81%	88,59%	88,59%	88,49%	89,75%

JavaScript	appium	90,70%	90,29%	89,65%	76,57%	90,06%
	meteor	60,82%	64,72%	62,23%	56,12%	62,94%
	node	70,82%	69,97%	73,99%	71,23%	72,36%
	pouchdb	75,29%	75,36%	76,29%	75,79%	76,80%
Python	kuma	89,42%	89,48%	89,22%	89,42%	89,32%
	pulp	94,16%	94,16%	93,97%	94,16%	93,53%
	rosdistro	94,32%	93,95%	94,08%	94,32%	94,21%
	scikit-learn	81,03%	81,00%	80,75%	81,07%	80,82%
Ruby	bundler	76,57%	76,31%	74,91%	91,05%	75,17%
	diaspora	77,14%	76,70%	77,32%	76,48%	76,61%
	metasploit Framework	85,86%	85,84%	86,10%	85,77%	85,77%
	nancy	84,41%	84,23%	83,60%	84,41%	83,96%
	vagrant	75,43%	77,13%	75,85%	76,07%	75,71%
Scala	marathon	87,40%	87,40%	87,49%	87,40%	84,87%
	dcala IDE	86,59%	86,04%	86,70%	86,59%	85,38%
	scala JS	94,51%	94,51%	94,29%	94,51%	94,73%
	Média	83,24%	82,99%	**83,51%**	82,90%	82,85%
	Média do ranking	2,75	3,025	**2,825**	2,975	3,425
	Número de vitórias	3	3	**7**	4	3

Fonte: Elaboração própria.

É importante salientar que o número de vitórias considera uma vitória para os dois algoritmos que por acaso apresentem um empate na primeira posição, é o caso do que ocorre para os algoritmos IBk e SMO nos projetos Nancy e Rosdistro. Na tabela é possível notar que a maior média de acurácia é 83,51% e pertence ao algoritmo *Random Forest* e que a menor média de *ranking* é igual a 2,75 e pertence ao algoritmo IBk. O algoritmo Random Forest foi considerado o melhor algoritmo para o contexto da pesquisa por possuir além da maior média de acurácia, 7 vitórias perante os demais algoritmos, sendo 4 a mais que o algoritmo IBk. Deste modo, considerou-se como padrão para os experimentos seguintes o algoritmo Random Forest com número de iterações igual a 400.

O algoritmo Random Forest também foi considerado o melhor para o contexto de previsão de aceitação de *pull request* em Gousios, Pinzger e Deursen (2014), o que confirma a indicação que esse algoritmo pode ser considerado como o mais adequado para esse contexto.

3.2.3 SELEÇÃO DE ATRIBUTOS

Tendo em vista que o processo de KDD é incremental e na tentativa de melhorar os resultados já obtidos. Buscou-se realizar um experimento comparativo utilizando estratégias de seleção de atributos. Neste experimento levou-se em conta a individualidade dos projetos, de modo que cada projeto seria constituído por um conjunto de atributos próprios e que melhor o representa.

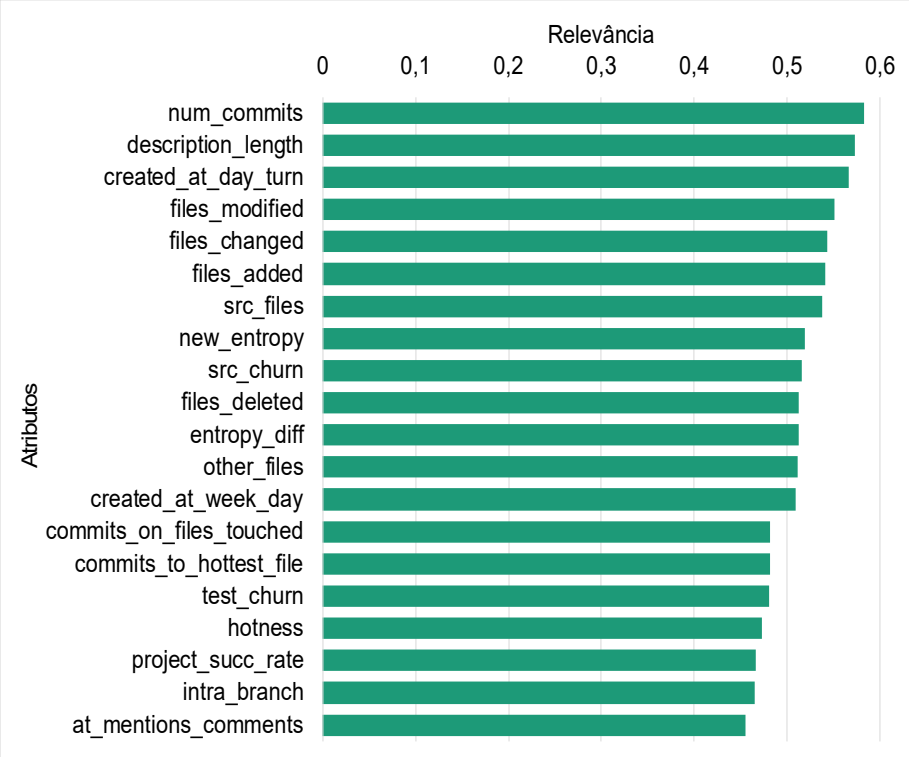
Duas estratégias foram exploradas para esse experimento, a primeiro consistiu em explorar a ferramenta de seleção de atributos disponibilizada pela Weka e a segunda explora a avaliação do algoritmo Random Forest que seleciona os atributos mais relevantes para gerar as árvores de decisão.

Na aba de seleção de atributos da ferramenta Weka foi selecionada a estratégia de seleção CfsSubsetEval. Esse tipo de algoritmo classifica os subconjuntos de atributos de acordo com medidas de avaliação de separabilidade (HALL, 1999). O resultado obtido em cada experimento foi a indicação de um subconjunto dos atributos mais correlacionados com a classe e menos correlacionados entre si. O subconjunto de atributos identificados como mais relevantes para cada projeto do conjunto A encontram-se na Tabela 8, a base de dados resultante constitui o conjunto D1.

A implementação do algoritmo Random Forest, possibilita a criação de um *ranking* dos atributos considerados mais relevantes para a identificação da classe. A Figura 10 trás um gráfico com os 20 atributos considerados, em média, como os mais relevantes dentre os 52 do conjunto A, os atributos são listados na vertical enquanto a relevância de cada um é representada na horizontal.

OS dois atributos mais relevantes e já presentes na base original são o número de *commits* (num_commits) e o tamanho da descrição do *pull request* (description_length). O terceiro atributo mais relevante foi o que indica o turno do dia no qual o *pull request* foi enviado (created_at_day_turn) um dos dois atributos propostos nesta pesquisa, o segundo atributo (created_at_week_day) obteve a décima terceira posição em média do ranking de relevância do algoritmo Random Forest.

Figura 10 - Relevância dos atributos para o algoritmo Random Forest



Fonte: Elaboração própria.

Como já falado buscou-se manter a especificidade de cada projeto, para isso foram selecionados os 15 atributos mais relevantes em cada projeto para constituir o conjunto D2. Para fins de comparação, os resultados do experimento são apresentados na Tabela 8, é possível perceber que o conjunto D2 possui maior média de acurácia (**81,73%**) e mais vitórias (15), enquanto o conjunto D1 apresentou a menor média do *ranking*.

Tabela 8 - Experimento comparativo entre os conjuntos D1 e D2

		Acurácia		
Linguagem	**Projeto**	**Conjunto D1**	**Conjunto D2**	**Número de atributos no conjunto D1**
Java	candlepin	87,16%	90,11%	6
	hazelcast	91,22%	91,73%	8
	netty	68,98%	57,78%	8
	okhttp	85,88%	88,01%	12
JavaScript	appium	89,24%	89,06%	16
	meteor	55,41%	57,45%	3
	node	71,79%	63,31%	9
	pouchdb	74,14%	76,36%	5
Python	kuma	89,61%	88,60%	4

	pulp	94,01%	94,01%	11
	rosdistro	93,34%	93,82%	5
	scikit-learn	79,78%	80,78%	2
Ruby	bundler	73,95%	75,35%	3
	diaspora	74,58%	75,24%	8
	metasploit Framework	79,31%	85,71%	4
	nancy	83,33%	83,60%	11
	vagrant	75,28%	75,71%	1
Scala	marathon	84,02%	86,98%	5
	dcala IDE	84,29%	86,59%	5
	scala JS	94,40%	94,29%	3
	Média	81,49%	**81,73%**	6,45
	Média do ranking	1,725	**1,275**	
	Número de vitórias	6	**15**	

Fonte: Elaboração própria.

Por possuir duas melhores métricas das três utilizadas, é possível afirmar que o conjunto D2 é ligeiramente melhor que o conjunto D1, sendo possível concluir que uma seleção mais específica como a do algoritmo Random Forest é mais eficiente para o contexto dessa pesquisa do que a aplicação do algoritmo de seleção CfsSubsetEval.

3.2.4 COMPARAÇÃO ENTRE AS ABORDAGENS

Dendo em vista as diferentes abordagens utilizadas neste trabalho buscou-se realizar um experimento comparativo que possibilitasse a verificação de desempenho de cada uma.

Foram obtidos ao longo desta pesquisa 5 conjuntos de dados originados da base de dados original:

a) **Conjunto A:** Constituído por 20 projetos e 52 atributos que remetem a informações disponíveis no momento de envio do *pull request;*
b) **Conjunto B:** Com 20 projetos e com os 16 atributos utilizados em Gousios, Pinzger e Deursen (2014);
c) **Conjunto C:** Utilizado para os experimentos de calibragem e composto por 5

projetos com os mesmos atributos do conjunto A;

d) **Conjuntos D1 e D2:** Resultantes da seleção de atributos e constituídos por 20 projetos, sendo que D1 apresenta um número variado de atributos para cada projeto e D2 apresenta 15 atributos que podem variar de projeto para projeto.

Em detrimento do conjunto C ter sido formado para a realização de calibragem e que o conjunto D2 apresentou resultados comparativos melhores do que o D1, realizou-se um experimento comparativo entre os conjuntos A, B e D2. Os resultados do experimento compõem a Tabela 9, as colunas apresentam a linguagem de programação de cada projeto, o nome do projeto e os valores de acurácia obtidos para os conjuntos A, B e D2. As métricas utilizadas na comparação foram: Média de acurácia, média do ranking e número de vitórias.

Tabela 9 - Experimento comparativo entre os conjuntos A, B e D2

		Acurácia		
Linguagem	**Projeto**	**Conjunto A**	**Conjunto B**	**Conjunto D2**
Java	Candlepin	91,05%	89,89%	90,11%
	Hazelcast	92,36%	92,11%	91,73%
	Netty	71,76%	70,57%	57,78%
	Okhttp	88,59%	87,52%	88,01%
JavaScript	Appium	89,65%	90,06%	89,06%
	Meteor	62,23%	60,37%	57,45%
	Node	73,99%	73,58%	63,31%
	Pouchdb	76,29%	78,09%	76,36%
Python	Kuma	89,22%	89,19%	88,60%
	Pulp	93,97%	93,92%	94,01%
	Rosdistro	94,08%	94,30%	93,82%
	Scikit Learn	80,75%	81,07%	80,78%
Ruby	Bundler	74,91%	73,16%	75,35%
	Diaspora	77,32%	78,07%	75,24%
	Metasploit Framework	86,10%	86,08%	85,71%
	Nancy	83,60%	84,86%	83,60%
	Vagrant	75,85%	75,71%	75,71%
Scala	Marathon	87,49%	85,88%	86,98%
	Scala IDE	86,70%	86,15%	86,59%
	Scala JS	94,29%	94,51%	94,29%
	Média da acurácia	**83,51%**	83,25%	81,73%

Média do ranking	**1,6**	1,975	2,425
Número de vitórias	**11**	7	2

Fonte: Elaboração própria.

Apesar das médias de acurácia serem muito semelhantes é possível perceber que o conjunto A se destaca com a maior média, 83,51%. Esse conjunto também apresenta menor média do *ranking* e maior número de vitórias.

O fato do conjunto D2 ter obtido os piores valores de métrica pode indicar que analisar atributos específicos para cada projeto pode não ser tão eficiente para determinar se o projeto deve ser aceito ou rejeitar *pull requests.* Isso demonstra a complexidade relacionada a essa decisão por parte do integrador, o que ressalta a grande quantidade de tempo empregada nesse processo de tomada de decisão.

Na comparação dos resultados do experimento comparativo o conjunto B, que apresenta os atributos utilizados em Gousios, Pinzger e Deursen (2014), apresentou o segundo melhor valor de média e número de vitórias, isso implica que apesar de ser uma abordagem eficiente, um conjunto de atributos mais robustos como os do conjunto A é mais eficiente, melhorando ligeiramente a acurácia em 0,99%. É importante salientar que essa melhora se refere ao contexto e abordagem construído ao longo deste trabalho e que a comparação ocorre especificamente entre os conjuntos de atributos e não quanto as abordagens de ambos os trabalhos.

4 CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES

Neste capítulo são apresentadas as considerações finais na subseção 4.1 e as recomendações para trabalhos futuros na subseção 4.2.

4.1 CONSIDERAÇÕES FINAIS

Este trabalho foi realizado em dois contextos mutualmente importantes. O primeiro consiste na análise do paradigma de desenvolvimento distribuído, mais precisamente para projetos *open-source* onde a contribuição externa através do modelo de *pull requests* auxilia na resolução de *bugs* e no desenvolvimento de novas funcionalidades. Deste modo, o trabalho contribui para a compreensão deste paradigma através da exploração das características do *pull request.*

O segundo contexto importante no qual esse trabalho se insere é o de mineração de dados, ressaltando características importantes desta área como o processo de KDD e os algoritmos de classificação.

Este trabalho também contribui para a consolidação do método *training-test sliding validation* como uma ferramenta importante para a mineração de dados em bases que possuem características temporais, especialmente nas tarefas preditivas.

As dificuldades encontradas na execução deste trabalho se concentraram na fase

inicial de seleção de atributos, mais precisamente na compreensão do significado de cada atributo, por vezes a utilização de atributos que não deveriam participar da etapa de treinamento do algoritmo de classificação, tais como o número de arquivos alterados depois do envio do *pull request* e o nome de cada um deles, conduziu a necessidade de refazer os experimentos para garantir a validade dos resultados.

Por fim, é possível afirmar que os experimentos cumpriram, dentro do contexto estabelecido, o objetivo de melhorar a taxa de acerto na previsão de integração de *pull request* através do cumprimento dos objetivos específicos estipulados, uma vez que se obteve uma melhora relativa de 0,99% na acurácia de conjuntos de dados de trabalhos anteriores.

Percebeu-se através da análise de relevância que os atributos referentes ao dia da semana e turno do dia do envio do *pull request* e que foram propostos por esse trabalho, foram importantes para o contexto, apresentando respectivamente a decima terceira e terceira posição no ranking de relevância do algoritmo Random Forest.

4.2 RECOMENDAÇÕES

Sugere-se como trabalho futuro o desenvolvimento de um *plugin* para navegador ou a implementação nos SCVD, que possibilite a coleta de atributos relativos a um *pull request* em tempo de execução e que com base nas informações obtidas sugira ao integrador a aceitação ou rejeição da solicitação, o que facilitaria esse processo.

Outro possível trabalho relacionado a área de previsão de integração de *pull request*, seria um analise mais extensiva para identificar e compreender os fatores que conduzem a rejeição de um *pull request* por parte de seu integrador, uma vez que esse contexto pode ser mais complexo de identificar.

Uma outra questão levantada por esse trabalho, e que em detrimento do seu escopo não foi totalmente explorada, é o significado dos atributos considerados mais significativos pelo algoritmo Random Forest para a aceitação ou rejeição dos *pull requests*, algo que poderia ser melhor explicado.

REFERÊNCIAS

BAR, Moshe; FOGEL, Karl. **Open source development with CVS**. 3. ed. Scottsdale, AZ: Paraglyph Press, 2003.

BARR, Earl T. et al. Cohesive and Isolated Development with Branches. In: DE LARA, Juan; ZISMAN, Andrea (Eds.). **Fundamental Approaches to Software Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 7212p. 316–331.

BIRD, Christian; ZIMMERMANN, Thomas. Assessing the value of branches with what-if analysis. In: PROCEEDINGS OF THE ACM SIGSOFT 20TH INTERNATIONAL SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING - FSE '12 2012, Cary, North Carolina. **Anais**... . In: THE ACM SIGSOFT 20TH INTERNATIONAL SYMPOSIUM. Cary, North Carolina: ACM Press, 2012. Acesso em: 2 set. 2019.

CAMILO, Cássio Oliveira. Mineração de Dados: Conceitos, Tarefas, Métodos e Ferramentas. [s. l.], p. 29, 2009.

CHACON, Scott; STRAUB, Ben. **Pro Git**. 2. ed. Berkely, CA, USA: Apress, 2014.

CIOS, Krzysztof J. et al. (EDS.). **Data mining: a knowledge discovery approach**. New York, NY: Springer, 2007.

DANTAS, Eric Rommel G.; PESSOA, João. O Uso da Descoberta de Conhecimento em Base de Dados para Apoiar a Tomada de Decisões. [s. l.], p. 10, 2008.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**, [s. l.], v. 17, n. 3, p. 37–37, 1996. a.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. The KDD process for extracting useful knowledge from volumes of data. **Communications of the ACM**, [s. l.], v. 39, n. 11, p. 27–34, 1996. b.

FILHO, Wilson. **Engenaria de Software**. 3. ed. Rio de Janeiro: LTC, 2012.

FRANK, Eibe; WITTEN, Ian H. Generating Accurate Rule Sets Without Global Optimization. In: IN: PROC. OF THE 15TH INT. CONFERENCE ON MACHINE LEARNING 1998, **Anais**... : Morgan Kaufmann, 1998.

GIL, Antonio. **Como Elaborar Projetos de Pesquisa**. 3. ed. São Paulo: Atlas, 1991.

GOLDSCHMIDT, Ronaldo; PASSOS, Emmanuel. **Data mining: um guia Prático**. 2. ed. Rio de Janeiro: Elsevier, 2005.

GOUSIOS, Georgios et al. Work Practices and Challenges in Pull-based Development: The Integrator's Perspective. In: 2015, Florence, Italy. **Anais**... . In: PROCEEDINGS OF THE 37TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE). Florence, Italy: IEE, 2015. Disponível em: <http://www.gousios.gr/pub/pullreqs-integrators.pdf#viewer.action=download>. Acesso em: 18 nov. 2019.

GOUSIOS, Georgios; PINZGER, Martin; DEURSEN, Arie Van. An exploratory study of the pull-based software development model. In: PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE 2014 2014, Hyderabad, India. **Anais**... . In: THE 36TH INTERNATIONAL CONFERENCE. Hyderabad, India: ACM Press, 2014. Acesso em: 2 set. 2019.

HALL, Mark A. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In: ICML 1999, **Anais**... [s.l: s.n.]

HAN, Jiawei; KAMBER, Miceline; PEI, Jian. **Data Mining: Concepts and Techniques**. 3. ed. EUA, Massachusetts, Waltham: Elsevier, 2012.

HAND, David; MANNILA, Heikki; SMYTH, Padhraic. **Principles of Data Mining**. Londres, Inglaterra: Massachusetts Institute of Technology, The Mit Press, 2001.

LIMA JÚNIOR, Manoel. **Previsão de Integradores e Tempo de Vida de Pull Requests**. 2017. Universidade Federal Fluminense, Niterói, 2017.

LIMA JÚNIOR, Manoel et al. Predicting the Lifetime of Pull Requests. **Journal Information and Software Technology**, [s. l.], p. 27, 2019.

LIMA, Max W. **Uma Extensão da Ferramenta Weka para Avaliação de Tarefas Preditivas**. 2017. Universidade Federal do Acre, Rio Branco, 2017.

MADDILA, Chandra; BANSAL, Chetan; NAGAPPAN, Nachiappan. 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 - ESEC/FSE 2019 2019, Tallinn, Estonia. **Anais**... . In: THE 2019 27TH ACM JOINT MEETING. Tallinn, Estonia: ACM Press, 2019. Acesso em: 2 set. 2019.

MURTA, Leonardo. **Gerência de Configuração no Desenvolvimento baseado em Componentes**. 2006. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

PLATT, John C. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. [s. l.], p. 21, 1998.

PRESSMAN, Roger. **Engenharia de Software**. São Paulo: Pearson Makron Books, 1995.

RODRIGUES, Alan. **Previsão da Naturez de Ocorrências Policiais na Cidade de Rio Branco**. 2018. Universidade Federal do Acre, Rio Branco, 2018.

SOARES, Daricélio. **On The Nature Of Pull Request: A Study About This Collabortion Paradigm Over Open-Source Projects Using Association Rules**. 2017. Universidade Federal Fluminense, Niterói, 2017.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

TRONCHONI, Alex B. et al. Descoberta de conhecimento em base de dados de eventos de desligamentos de empresas de distribuição. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, [s. l.], v. 21, n. 2, p. 185–200, 2010.

WAZLAWICK, Raul. **Metodologia de Pesquisa para Ciência da Computação**. Rio de Janeiro: Elsevier, 2009.

WITTEN, Ian H.; FRANK, Eibe; HALL, Mark A. **Data mining: practical machine learning tools and techniques**. Third edition ed. Amsterdam: Morgan Kaufmann, 2011.

YU, Yue et al. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In: 2015 IEEE/ACM 12TH WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES 2015, Florence, Italy. **Anais**... . In: 2015 IEEE/ACM 12TH WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES (MSR). Florence, Italy: IEEE, 2015. Disponível em: <http://ieeexplore.ieee.org/document/7180096/>. Acesso em: 29 out. 2019.

ZHOU, Zhi-Hua. Three perspectives of data mining. **Artificial Intelligence**, [s. l.], v. 143, n. 1, p. 139–146, 2003.