UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EUCLIDES RAMOS DE ARAÚJO FILHO

INVESTIGANDO BUGS REABERTOS: UM ESTUDO DE CASO NO BUGZILLA

CAMPINA GRANDE - PB

2022

EUCLIDES RAMOS DE ARAÚJO FILHO

INVESTIGANDO BUGS REABERTOS: UM ESTUDO DE CASO NO BUGZILLA

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Franklin de Souza Ramalho

CAMPINA GRANDE - PB

2022

EUCLIDES RAMOS DE ARAÚJO FILHO

INVESTIGANDO BUGS REABERTOS: UM ESTUDO DE CASO NO BUGZILLA

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Franklin de Souza Ramalho

Orientador – UASC/CEEI/UFCG

Everton Leandro Galdino Alves

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de Setembro de 2022.

CAMPINA GRANDE - PB

RESUMO

Dentre as atividades típicas de um processo de software, podemos destacar as tarefas de testar, analisar, reportar e corrigir bugs. A realização dessas tarefas é importante para identificar erros comuns ou complexos durante todas as etapas do desenvolvimento, evitando retrabalho e entregando um software com mais qualidade e confiabilidade [1]. Em um relatório de bug, geralmente, seu autor oferece detalhes da anormalidade que vem ocorrendo. Tipicamente, um relatório de bug é aberto, o bug é corrigido e o relatório é fechado. Contudo, por vezes, é verificado que a correção do bug não foi eficaz, seja por falta de descrição mais objetiva no relatório, seja por dificuldade de entendimento por parte do desenvolvedor. Assim, faz-se necessária a reabertura do bug, adicionando tempo no processo de desenvolvimento, tornando o software mais custoso. Por isso, é importante investigar o que pode ser feito para mitigar tais problemas. Neste trabalho, investigamos as características que levam um bug a ser reaberto. Os resultados deste trabalho podem ajudar aos usuários finais e desenvolvedores a melhor escrever relatórios de bugs, bem como aos desenvolvedores a melhor entendê-los e tratá-los. O estudo utilizou um dataset extraído da ferramenta Bugzilla.

Investigando Bugs Reabertos: Um Estudo de Caso no Bugzilla

Euclides Ramos de Araújo Filho
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
euclides.filho@ccc.ufcg.edu.br

Franklin Ramalho
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
franklin@computacao.ufcg.edu.br

RESUMO

Dentre as atividades típicas de um processo de software, podemos destacar as tarefas de testar, analisar, reportar e corrigir bugs. A realização dessas tarefas é importante para identificar erros comuns ou complexos durante todas as etapas do desenvolvimento, evitando retrabalho e entregando um software com mais qualidade e confiabilidade [1]. Em um relatório de bug, geralmente, seu autor oferece detalhes da anormalidade que vem ocorrendo. Tipicamente, um relatório de bug é aberto, o bug é corrigido e o relatório é fechado. Contudo, por vezes, é verificado que a correção do bug não foi eficaz, seja por falta de descrição mais objetiva no relatório, seja por dificuldade de entendimento por parte do desenvolvedor. Assim, faz-se necessária a reabertura do bug, adicionando tempo no processo de desenvolvimento, tornando o software mais custoso. Por isso, é importante investigar o que pode ser feito para mitigar tais problemas. Neste trabalho, investigamos as características que levam um bug a ser reaberto. Os resultados deste trabalho podem ajudar aos usuários finais e desenvolvedores a melhor escrever relatórios de bugs, bem como aos desenvolvedores a melhor entendê-los e tratá-los. O estudo utilizou um dataset extraído da ferramenta Bugzilla.

Palavras-chave

Defeito, Processo de Software, Reabertura de bug, Relatório de bug.

ABSTRACT

Among the typical activities of a software process, we can highlight the tasks of testing, analyzing, reporting and fixing bugs. Performing these tasks is important to identify common or complex errors during all stages of development, avoiding rework and delivering software with more quality and reliability [1]. In a bug report, the author usually provides details of the abnormality that has been occurring. Typically, a bug report is opened, the bug is fixed, and the report is closed. However, sometimes it is verified that the bug correction was not effective, either due to lack of a more objective description in the report, or due to difficulty in understanding on the part of the developer. Thus, it is necessary to reopen the bug, adding time to the development process, making the software more expensive. Therefore, it is important to investigate what can be done to mitigate such problems. In this work, we investigate the characteristics that lead a bug to be reopened. The results of this work can help end users and developers to better write bug reports, as well as developers to better understand and handle them. The study used a dataset extracted from the Bugzilla tool.

Keywords

Bug, Software Process, Bug reopening, Bug report.

1. INTRODUÇÃO

Em um processo de desenvolvimento de software, existem atividades que buscam testar o software, analisar, reportar e corrigir bugs, a fim de alcançar um software de maior qualidade. Geralmente, bugs são identificados dentro do software e, para tal, cria-se um relatório para que a equipe de desenvolvimento, responsável pelo software, tome conhecimento do comportamento indesejado que está ocorrendo e, assim, consiga solucionar tal problema. Entretanto, algumas vezes, a solução aplicada não resolve o problema, fazendo com que o relatório de bug seja reaberto. Isso acontece por diversos motivos, sobretudo pela falta de artefatos ou informações suficientes para identificar o bug ou pela dificuldade de entendimento por parte do desenvolvedor a quem foi atribuída a tarefa de solucionar o problema.

Pesquisas já identificaram que bugs descobertos por usuários [2] e bugs em componentes mais complexos tendem à reabertura [3]. Devido a tal eventualidade, o desenvolvimento do software tende a demorar mais para ser finalizado, fazendo com que um custo adicional seja gerado em cima disso e, nos softwares em produção, causa impactos relevantes, como por exemplo, no mau funcionamento de determinadas funcionalidades dentro do software e, até mesmo, abrindo margem para problemas de segurança, como aconteceu com o Facebook [4]. Com esses problemas relacionados à reabertura do bug, surgem lacunas no desenvolvimento que acarretam problemas a curto e longo prazo.

Este trabalho tem como principal objetivo investigar e identificar o que acontece em um relatório de bug, que faz com que o mesmo seja reaberto, e o que pode ser feito ou aperfeiçoado para que tal dificuldade seja mitigada ou minimizada. O impacto deste trabalho se dá pela importância e relevância deste tema na atualidade, impactando diretamente no processo de desenvolvimento de software, mais especificamente no processo de reportar e corrigir bugs, diminuindo o tempo perdido em bugs que foram reabertos.

Para a investigação, foi utilizado um dataset, extraído da ferramenta Bugzilla [5]. O dataset foi extraído utilizando uma configuração específica de filtragem de busca no Bugzilla, fazendo com que apenas bugs com status *resolved,* e que tenha tido o status *reopened,* estejam inseridos dentro do dataset. Com o

uso do *dataset*, foram executados experimentos, utilizando modelos de classificação de dados, avaliados através de métricas de desempenho, como acurácia, precisão, cobertura e f1-score, identificando *features* promissoras para a obtenção dos resultados.

Este documento está organizado como segue. A Seção 2 faz uma breve introdução de conceitos importantes para o melhor entendimento deste trabalho. A Seção 3 detalha os métodos utilizados para a obtenção dos resultados. A Seção 4 descreve os resultados obtidos, com gráficos e tabelas obtidas. A seção 5 apresenta as ameaças à validade dos resultados da pesquisa. A Seção 6 indica alguns trabalhos relacionados à pesquisa. A Seção 7 descreve as conclusões obtidas através da pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão introduzidos alguns conceitos, essenciais para o entendimento detalhado do estudo, são eles: detalhamento da ferramenta de busca utilizada para obtenção dos bugs, Bugzilla, conceitos de aprendizagem de máquina, classificação, validação e avaliação de modelos.

2.1 Bugzilla

A ferramenta Bugzilla é uma ferramenta bastante robusta, baseada em web, que dá suporte para desenvolvedores rastrearem e acompanharem bugs, problemas, questões, melhorias e outras solicitações de alterações pendentes em seus produtos de forma eficaz [5]. Dentro dela, é possível cadastrar novos bugs ou contribuir em bugs já existentes (realizando criação de conta e login). Dentro da página de busca avançada, o usuário consegue filtrar o bug por diversas características: *Classification, Product, Component, Status* (identifica em qual situação o bug se encontra: aberto, resolvido, reaberto, etc)*, Resolution* (identifica qual o tipo de solução aplicada para resolver o bug: corrigido, inválido, duplicado, incompleto, não é um bug, etc)*, Version* (identifica a versão do software em que o bug foi encontrado)*, Target Milestone, Type* (identifica o tipo do bug: melhoria, defeito ou uma tarefa)*, Severity* (identifica o nível de impacto que o bug causa)*, Priority* (identifica o nível de prioridade em que o bug deve ser corrigido)*, Hardware* (identifica em qual tipo de hardware o bug foi encontrado)*, OS* (identifica em qual sistema operacional o bug foi encontrado). Além disso, é possível realizar a busca por campos específicos como *status*, *attachments* (relacionado aos arquivos anexados ao bug)*, cc* (relacionado às pessoas interessadas no bug), *comment (*relacionado aos comentários inseridos no bug)*, depends on, duplicates, regressed by,* entre outros [6].

2.2 Classificação

Aprendizado de Máquina é uma área de pesquisa da Inteligência Artificial que visa ao desenvolvimento de programas de computador com a capacidade de aprender a executar uma determinada tarefa com sua própria experiência [7]. Na aprendizagem de máquina, é possível treinar computadores, através de algoritmos, para predizer algo, através do reconhecimento de padrões aprendidos no treinamento. Dentro dessa área, existem três subáreas principais: Classificação; Regressão; e Agrupamento [8]. Este trabalho aplica algoritmos de classificação.

Na classificação, espera-se que o modelo preveja a categoria de uma observação dada [8]. Isso é feito através de associações de conjuntos de observação sobre a mesma caracterização [9]. O treinamento do modelo de classificação dos dados pode ser feito utilizando dois tipos de aprendizado: supervisionado e não supervisionado. No processo supervisionado, utiliza-se dados já observados que possuem classificações rotuladas, permitindo comparar as previsões com os valores reais. No processo não supervisionado, não se utiliza dados já observados para prever dados reais, a técnica utilizada é agrupar dados com comportamento similar entre eles.

Para a pesquisa, foram escolhidos os seguintes classificadores:

- *Naive Bayes*: tem um bom desempenho quando submetido a variáveis categóricas e, supondo que haja independência entre os atributos, o desempenho desse modelo é superior a outros modelos mais complexos. Esse modelo utiliza o aprendizado supervisionado [10]. Ele foi escolhido para a pesquisa dada a sua característica de desconsiderar relação entre as *features*, assim, podendo identificar se, de fato, não há essa relação;
- *Decision Tree*: faz uso de aprendizado supervisionado e suas principais vantagens são a facilidade na compreensão do algoritmo a relações não-lineares entre os parâmetros não afetam o desempenho do modelo [11]. Este algoritmo foi escolhido para a pesquisa pois, dado relações não-lineares entre os parâmetros, o desempenho não é afetado;
- *K-Nearest Neighbor (KNN)*: possui um bom desempenho quando os dados possuem relacionamento complexo entre características, além de não requerer muitos ajustes nos parâmetros para otimizar o desempenho do algoritmo. Esse algoritmo utiliza aprendizagem supervisionada [12]. Como este algoritmo possui bom desempenho quando os dados possuem relacionamento complexo entre características, ele também foi escolhido para esta pesquisa.

A seguir, são apresentados e descritos os classificadores utilizados.

2.2.1 Naive Bayes

O *Naive Bayes* [13], ou Classificador de Bayes Ingênuo, é um classificador probabilístico baseado no Teorema de Bayes que desconsidera qualquer tipo de relação entre as *features*. Ele consiste em encontrar uma probabilidade que um evento aconteça a partir de outro evento que pode ter relação com ele [13]. Na Eq. 1 [13], podemos visualizar a definição da probabilidade de um evento A ocorrer, dado que o evento B ocorra, definido pelo Teorema de Bayes.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \quad (1)$$

Nesta pesquisa, podemos ilustrar da seguinte maneira: dado os valores visíveis na Tabela 1, o cálculo da probabilidade no treino é calculado como disposto na Tabela 2. Para as colunas “Valor/Ocorrência”, “Chance de Status ser 1 - 3/6” e “Chance de Status ser 0 - 3/6”, o valor é apresentado como quantidade de vezes que o valor aparece para a *feature* / quantidade da *feature*. Quando o valor não aparece para aquela *feature*, a célula correspondente à chance daquele evento ocorrer fica 0. Quando todos os valores da *feature* são iguais ao comparado, o valor da chance daquele evento é 1. Quanto mais perto de 1, maior a chance do evento ocorrer. Na coluna de Status, 1 representa bug reaberto e 0 representa bug não reaberto.

Quantidade de Comentários	Quantidade de CC	Quantidade de Arquivos	Status
5	7	3	0
15	10	5	1
20	7	3	1
7	3	5	0
5	3	3	0
20	7	10	1

Tabela 1: Ilustração de exemplos de dados de bugs.

	Valor/Ocorrência	Chance de Status ser 1 3/6	Chance de Status ser 0 3/6
Quantidade de Comentários	5/2	0	1/1
	7/1	0	1/1
	15/1	1/1	0
	20/2	1/1	0
Quantidade de CC	3/2	0	100
	7/3	2/3	1/3
	10/1	1/1	0
Quantidade de Arquivos	3/3	1/3	2/3
	5/2	1/2	1/2
	10/1	1/1	0

Tabela 2: Exemplo de como é feito o cálculo da probabilidade de um evento ocorrer.

2.2.2 Decision Tree

O Classificador Árvore de Decisão, ou *Decision Tree* [14], é uma estrutura hierárquica de nós, com um conjunto de escolhas, que resulta em uma decisão final. A sua estrutura consiste em nodos (nós), que representam os atributos, e arcos (ramos), provenientes desses nodos, e que recebem os valores possíveis para esses atributos. Nas árvores existem nodos folha (folha da árvore), que representam as diferentes classes de um conjunto de treinamento, ou seja, cada folha está associada a uma classe. Cada percurso na árvore (da raiz à folha) corresponde a uma regra de classificação [14][15].

Esse classificador, *Decision Tree,* utiliza a estratégia dividir para conquistar, fazendo com que os dados do problema sejam divididos em vários subconjuntos, agrupando-os por características semelhantes, e, por sua vez, esse vários subconjuntos são divididos em outros vários subconjuntos [14]. Na Figura 1, podemos visualizar a representação de um classificador baseado em árvore de decisão, inserido do contexto da pesquisa, utilizando os mesmos dados da Tabela 1 (levando em consideração apenas a coluna "Quantidade de Comentários"), aplicados no treinamento do modelo. A partir do valor do atributo "Quantidade de Comentários", o classificador define se o bug será reaberto ou não. O nó raiz (Quantidade de Comentários) é a feature que está sendo analisada para responder a pergunta "O bug é reaberto?". Seus arcos (5, 7, 15 e 20) representam os valores possíveis dos dados utilizados. E as folhas (SIM e NÃO) representam a resposta para a pergunta em questão, de acordo para cada valor a ser analisado.

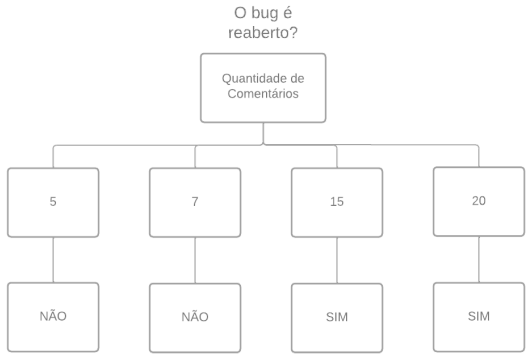

Figura 1: Exemplo de um classificador utilizando árvore de decisão.

2.2.3 K-Nearest Neighbor (KNN)

O algoritmo *K-Nearest Neighbor* [16], ou KNN ou K-Vizinhos Mais Próximos, é um algoritmo de aprendizagem supervisionado do tipo *lazy*, ou algoritmo preguiçoso (não necessita de dados de treinamento para gerar o modelo), introduzido por Aha, e consiste em encontrar os *k* exemplos rotulados mais próximos do exemplo não classificado, e, como base no rótulo desse exemplo encontrado, é definida uma classe para o exemplo não rotulado [16]. Essa proximidade é dada através do uso de alguma métrica (distância). Isto é, dados dois pontos em um espaço n-dimensional $X = (x_1, x_2, \dots, x_n)$ e $Y = (y_1, y_2, \dots, y_n)$, a distância entre eles pode ser dada por [17]:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$ - Métrica Euclidiana

$$d(X, Y) = \sum_{i=1}^{n} |x_i - y_i|$$ - Métrica de Manhatan

$$d(X, Y) = \sqrt[q]{\sum_{i=1}^{n} (x_i - y_i)^q}$$ - Métrica de Minkowski

Sendo que a mais comum é a métrica Euclidiana [17].

Para a execução do algoritmo KNN, três parâmetros devem ser determinados: quais exemplos rotulados devem ser

lembrados, qual a medida que quantifica a similaridade entre o exemplo não classificado e os exemplos de treinamento e quantos vizinhos mais próximos devem ser considerados [16]. Na Figura 2, é ilustrado um exemplo de aplicação do algoritmo KNN, com um conjunto de treinamento descrito por dois atributos: (+), para bugs reabertos, e (-), para bugs não reabertos. O atributo com (?) será o atributo a ser rotulado. Utilizando k = 1, no exemplo da figura, a rotulagem resultante para (?) seria (+), pois o k vizinho encontrado possui rotulação (+).

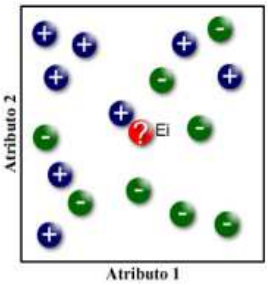

Figura 2: Exemplo de classificação do método KNN [16].

2.3 Métricas de Desempenho

Para realizar a medição assertiva das previsões realizadas pelos classificadores, existem algumas métricas muito importantes para a obtenção da validação dos classificadores. Existem diferentes métricas para avalizar o modelo e algumas funcionam melhor para um determinado problema. Nos modelos de classificação, a maioria das métricas são baseadas na matriz de confusão [18].

2.3.1 Matriz de Confusão

Uma Matriz de confusão apresenta todos os acertos e erros do modelo ao prever classes [18]. Essa informação é bastante importante para medir o desempenho do modelo. Ela é uma matriz quadrada em que se compara os verdadeiros valores de uma classificação com os valores preditos pelo modelo. Sua diagonal é composta pelos acertos do modelo e os demais valores indicam os erros cometidos [18]. A seguir, na Figura 3, podemos ver um exemplo de uma matriz de confusão.

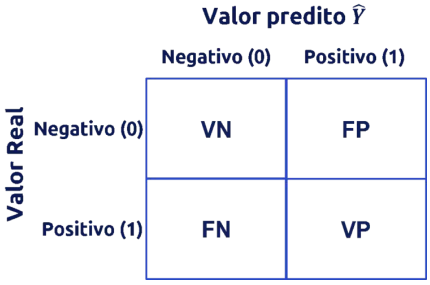

Figura 3: Matriz de confusão [18].

Cada quadrado da matriz apresenta a frequência de acertos e erros. Nos quadrados VN (Verdadeiro Negativo) e VP (Verdadeiro Positivo), fica a quantidade de acertos que o modelo previu, com base nos valores reais. Já nos quadrados FN (Falso Negativo) e FP (Falso Positivo), fica a quantidade de erros que o modelo previu, com base nos valores reais. O quadrado VN indica os valores que, no modelo real, são negativos e foram classificados corretamente como negativos. O quadrado VP indica os valores que, no modelo real, são positivos e foram classificados corretamente como positivos. Já o quadrado FN indica os valores positivos, no modelo real, classificados como negativos. O quadrado FP indica os valores negativos, no modelo real, classificados como positivos. A partir desses valores, podem ser calculadas métricas de desempenho. As mais comuns são: acurácia, precisão, cobertura e F1-Score.

2.3.1.1 Acurácia

A Acurácia indica o número de previsões corretas, apresentando uma performance geral do modelo [18]. Essa métrica é calculada através da somatória de acertos nas diferentes classes (VN + VP) dividido pelo número total de registros, como visto na Eq. 2 [18].

$$Acurácia = \frac{VN + VP}{VN + VP + FN + FP} \quad (2)$$

2.3.1.2 Precisão

A Precisão indica o número de previsões classificadas como positivas e que realmente eram positivas [18]. Essa métrica é calculada através da divisão entre os verdadeiros positivos (VP) e o total de positivos classificados pelo modelo (VP + FP). A Eq. 3 [18] demonstra como esse cálculo é feito.

$$Precisão = \frac{VP}{VP + FP} \quad (3)$$

2.3.1.3 Cobertura

A Cobertura, ou *recall* ou sensibilidade, indica a porcentagem dos Verdadeiros Positivos dentre todas as classificações que realmente são positivas no conjunto de dados [18]. O cálculo dessa métrica é dado através da divisão entre os verdadeiros positivos (VP) e a soma dos positivos do conjunto de dados (VP + FN), como podemos visualizar na Eq. 4 [18].

$$Cobertura = \frac{VP}{VP + FN} \quad (4)$$

2.3.1.4 F1-Score

A F1-Score indica uma média harmônica ponderada entre as métricas de precisão e cobertura [18]. Ela é muito utilizada quando há classes desbalanceadas, pois apresenta um valor mais fiel do modelo, tendo em vista que, nesses casos de classes desbalanceadas, o modelo pode ter boa cobertura e boa precisão. O cálculo dessa métrica pode ser visto na Eq. 5 [18].

$$F1Score = \frac{2 * precisão * cobertura}{precisão + cobertura} \quad (5)$$

3. METODOLOGIA

A metodologia utilizada nesta pesquisa se deu por meio da realização de estudo utilizando um *dataset* contendo bugs com status *resolved fixed* e que, em algum momento do seu ciclo de vida, teve status igual a *reopened* e que estejam dentro de um período determinado de tempo pré-estabelecido. O *dataset* foi extraído da ferramenta Bugzilla, sobre o qual foram realizadas análises qualitativas e quantitativas. O dataset escolhido visa atender alguns requisitos preestabelecidos para a pesquisa: o bug já deve ter sido resolvido; o bug foi reaberto durante seu ciclo de vida; o status do bug foi alterado dentro do período de 10 anos (01/01/2012 a 01/01/2022). Assim, será possível alcançar resultados que reflitam a realidade e que foque nos bugs de interesse do estudo.

A questão de pesquisa que este trabalho tenta responder é:

- QP1: Quais fatores levam a reabertura de um bug?

Para a obtenção dos resultados, foram realizados os seguintes passos: (i) seleção dos bugs de interesse e que atendam aos requisitos pré-estabelecidos; (ii) análise exploratória; (iii) seleção das *features* utilizadas no experimento; (iv) padronização nos dados faltosos dos bugs; (v) execução do experimento; (vi) análise dos resultados. Os artefatos utilizados na pesquisa estão disponíveis em um *Google Colaboratory*[1]. A seguir, cada etapa acima indicada será explicada e detalhada.

3.1 Seleção dos bugs

A primeira etapa do estudo foi a preparação do dataset, com a configuração específica de filtragem no Bugzilla, para capturar apenas bugs que se enquadram na pesquisa. A configuração de busca avançada no Bugzilla foi: status = RESOLVED (bugs já resolvidos); resolution = FIXED (bugs que foram corrigidos de fato); type = defect (bugs que foram classificados como defeito); search by change history = status changed to REOPENED between 2012-01-01 - 2022-01-01 (filtrando os bugs que tiveram o status alterado para reaberto dentro desse período de dez anos). Com isso, os dados obtidos foram salvos localmente, em arquivos do tipo Comma-separated values (CSV), para treinamento e teste dos modelos de classificação. Além desses bugs, também foram utilizados bugs que não foram reabertos, durante o seu ciclo de vida, realizando uma outra busca e concatenado ao *dataset*, para fins de comparação. Assim, o *dataset* totalizou um total de 20282 bugs (10000 bugs não reabertos e 10282 bugs reabertos).

3.2 Análise exploratória

Para compreendermos melhor o comportamento dos bugs e analisar características comuns dentro do processo de desenvolvimento de software, mais especificamente nos relatórios de bugs escritos por desenvolvedores e/ou usuários finais, partimos, inicialmente, da investigação de como se dá a incidência de reabertura dos bugs. Assim, se fez necessário realizar um estudo de caso e fazer um levantamento do comportamento dos bugs do *dataset* no que se diz respeito à quantidade de vezes que o mesmo bug foi reaberto. Utilizou-se a informação contida no histórico do bug que identifica quando houve alteração em seu status, mais especificamente, quando houve alteração para o status *REOPENED*. Com essa informação em mãos, foi possível contabilizar, pela quantidade de recorrência da alteração, a quantidade de vezes que o bug foi reaberto.

3.3 Seleção das *Features*

Para a execução do experimento, foi necessário realizar a verificação do que cada *feature* significa para o bug e quais influências ela pode trazer para a resolução do mesmo, a fim de definir quais *features* possuem relevância para a pesquisa. A partir do estudo sobre o significado e relação de cada *feature* com o bug, foi possível definir as *features* candidatas:

- **Number of CC:** quantidade de usuários que não podem ter um papel a desempenhar no bug, como desenvolvedor ou testador, por exemplo, mas que estão interessados em seu progresso. Essa informação pode ser relevante para a reabertura do bug ou não;
- **Number of Comments:** quantidade de comentários adicionados por usuários do Bugzilla para determinado bug. Pode estar relacionado à reabertura do bug;
- **Number of Attachments:** quantidade de arquivos anexados no bug, que são utilizados como evidência para uma melhor compreensão do comportamento. Dado à sua importância para compreensão do bug, essa *feature* pode ter relação com a reabertura do mesmo;
- **Number of Depends on:** quantidade de bugs que devem ser resolvidos antes que o bug possa ser resolvido. Por essa *feature* apresentar os relacionamentos de dependência entre bugs, pode ter relevância para a pesquisa;
- **Number of Regressed by:** quantidade de bugs que causaram a criação e inserção do bug. Também, por apresentar o relacionamento de dependência entre bugs, pode ser relevante para a pesquisa;
- **Number of Blocks:** quantidade de bugs que dependem que o bug seja resolvido para serem resolvidos. Também, por apresentar o relacionamento de dependência entre bugs, pode ser relevante para a pesquisa;
- **Flags** (em específico, se existe a flag *needinfo*): lista de flags adicionadas no bug e que representam algum tipo de informação para o bug. Neste caso, a flag *needinfo*, representa que o bug necessita de mais informações para ser compreendido e, assim, ser resolvido. Sendo assim, apresenta interesse para a pesquisa, pois apresenta uma informação importante do bug;
- **Changed:** identifica a última vez que o bug sofreu atualizações de qualquer tipo. Através dessa informação, poderemos mapear, utilizando o histórico de modificações do bug, qual foi a mudança no bug;
- **Creation date:** data de criação do bug. Importante para calcularmos o tempo de vida do bug;
- **Priority:** descreve a importância e a ordem em que um bug deve ser corrigido em comparação com outros bugs. Utilizado para definir a ordem de prioridade para correção de bugs. As prioridades possíveis são -- (vazio), P1, P2, P3, P4 e P5 [19]. Importante para mapearmos a importância dos bugs a ser classificado;
- **Severity:** descreve o impacto do bug. Os valores possíveis de severidade são S1, S2, S3, S4 e outros valores (podem ser criados valores específicos para severidade) [20];

[1] https://colab.research.google.com/drive/1u-5fE34Slcfo8TNQUf7Kc-n7zPF6wW99?usp=sharing

Também foi possível identificar as *features* que não serviriam para a análise da pesquisa:

- **Alias, Bug ID:** utilizados apenas para fins de identificação do bug;
- **Assignee:** identifica o encarregado de resolver o bug, não trazendo informações sobre o processo de resolução e por quais motivos o bug foi reaberto;
- **Classification:** utilizado para categorizar o bug em classificação, produtos e componentes, sem relação com a reabertura de bugs. A classificação é a categorização de nível superior, usada para agrupar vários produtos relacionados em uma entidade distinta. Por exemplo, uma empresa que fabrica jogos pode ter uma classificação de "Jogos" e um produto separado para cada jogo [24];
- **Comment Tag:** utilizado apenas para facilitar a filtragem dos comentários;
- **Component:** categoria de segundo nível. Cada um pertence a um produto. Também não é relevante para a pesquisa;
- **Duplicates:** Lista de bugs que foram marcados como uma duplicata do bug que está sendo visualizado no momento. Utilizado para identificar bugs duplicados;
- **Hardware:** identifica a plataforma de hardware em que o bug foi encontrado, irrelevante para fins de comparação sobre reabertura do bug;
- **Keywords:** vocabulário controlado para caracterizar bugs em produtos e componentes. Sem relação com reabertura de bugs;
- **Mentors:** usuários que se ofereceram para ajudar o dono do bug com tarefas relacionadas ao bug (*feature* pouco utilizada);
- **OS:** informa em qual sistema operacional o bug foi encontrado;
- **Product:** utilizado para categorizar bugs, juntamente com Classification e Component. Representa produtos de envio do mundo real. Por exemplo, uma empresa que fabrica jogos pode ter vários jogos (Tetris, Mario, etc) que são seus produtos [24]. Também não é relevante para a pesquisa;
- **QA Contact:** identifica o encarregado de confirmar o bug e verificar a correção assim que o bug for resolvido. Fugiria do escopo da pesquisa caso incluíssemos ela;
- **Rank:** Usado por alguns grupos para fornecer ordenação mais refinada para trabalhar em bugs do que o permitido pelo campo Prioridade. O uso deste campo é restrito ao grupo `rank-setters`. Assim, não contribuindo para a pesquisa;
- **Reporter:** identifica a pessoa que registrou o bug. Informação irrelevante para a pesquisa;
- **Summary:** descreve sucintamente do que se trata o bug.

Com essa definição de *features*, será possível analisar, mais detalhadamente, o comportamento do bug e o que pode influenciar para a reabertura do mesmo.

3.4 Padronização nos dados do bug

No *dataset* obtido, algumas colunas de dados possuíam células em branco e/ou possuíam valores diferentes do tipo inteiro, fazendo com que alguns modelos não levassem em consideração a coluna. Assim, se fez necessário realizar uma normalização dos dados, para que não houvesse inconsistência nos resultados.

Na coluna de *flags*, foi aplicada a normalização, fazendo com que, onde houvesse a tag buscada (*needinfo)*, o valor da célula fosse substituído por 1. Caso contrário, o valor da célula foi substituído por 0.

Na coluna de *Priority*, a normalização aplicada foi seguindo os níveis de prioridade definidos na documentação do Firefox [19] e definido o valor entre 0 e 5 para cada prioridade, como podemos ver na Tabela 3.

Prioridade	Valor normalizado
--	0
P1	1
P2	2
P3	3
P4	4
P5	5

Tabela 3: Tipos de prioridade e seus valores normalizados.

Na coluna de *Severity*, também foi aplicada uma normalização, similar à coluna de *Priority*, também seguindo os nomes estabelecidos no Bugzilla [20], conforme descrito na Tabela 4.

Severidade	Valor normalizado
S1	1
S2	2
S3	3
S4	4
Outros valores	0

Tabela 4: Tipos de severidade e seus valores normalizados.

Também, para diferenciar os bugs reabertos dos bugs não reabertos, foi inserida a coluna reopened, com valores 1 e 0, para os bugs reabertos e não reabertos, respectivamente.

3.5 Execução do experimento

Para a execução do experimento, foram utilizados três modelos de classificação: *Naive Bayes, Decision Tree* e *K-Nearest Neighbor* (KNN), todos provindos da biblioteca *scikit-learn* [21]. No processo de treinamento dos modelos, o *dataset* foi dividido aleatoriamente em 80% para treino e 20% para teste.

4. RESULTADOS

4.1 Análise Exploratória

Na análise exploratória, obtivemos o seguinte resultado, observado na Figura 4.

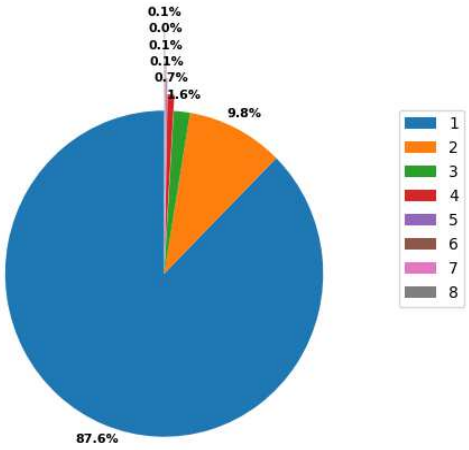

Figura 4: Quantidade de vezes que o bug foi reaberto. Azul: 1 vez; Laranja: 2 vezes; Verde: 3 vezes; Vermelho: 4 vezes; Lilás: 5 vezes; Marrom: 6 vezes; Rosa: 7 vezes; Cinza: 8 vezes.

Analisando o gráfico gerado através do *dataset,* com um total de 10282 bugs, e levando em consideração a recorrência de reabertura do bug, realizando a checagem através do histórico de modificação do bug, podemos verificar que 87,6% deles foram reabertos apenas uma única vez e que, assim, 12,4% foram reabertos duas ou mais vezes. Isso quer dizer que, durante o ciclo de vida da maioria desses bugs, o status dele foi mudado para *REOPENED* apenas uma vez. A Tabela 5 detalha o agrupamento dos bugs reabertos de acordo com o período (coluna) e a quantidade de vezes que o mesmo foi reaberto (linha).

Quantidade de vezes que o bug foi reaberto	Período					
	2012 a 2014	2014 a 2016	2016 a 2018	2018 a 2020	2020 a 2022	Total
1	2404 (23, 38%)	2414 (23, 47%)	1985 (19, 30%)	1236 (12, 02%)	963 (9, 36%)	9002
2	267 (2, 59%)	231 (2, 24%)	179 (1, 74%)	185 (1, 79%)	150 (1, 45%)	1012
3	35 (0, 34%)	26 (0, 25%)	20 (0, 19%)	48 (0, 46%)	40 (0, 38%)	169
4	5 (0, 04%)	6 (0, 05%)	13 (0, 12%)	22 (0, 21%)	25 (0, 24%)	71
5	3 (0, 02%)	2 (0, 01%)	0 (0%)	5 (0, 04%)	1 (0,00 9%)	11
6	1 (0.00 9%)	0 (0%)	0 (0%0	3 (0, 02%)	3 (0, 02%)	7
7	0 (0%)	0 (0%)	0 (0%)	2 (0, 01%)	2 (0, 01%)	4
8	0 (0%)	0 (0%)	0 (0%)	3 (0, 02%)	3 (0, 02%)	6
Total	2715 (26, 40%)	2679 (26, 05%)	2197 (21, 36%)	1504 (14, 62%)	1187 (11, 54%)	10282 (100%)

Tabela 5: Agrupamento dos bugs por período e pela quantidade que foi reaberto.

Na Tabela 5, podemos perceber que, conforme os anos avançam, a quantidade de bugs reabertos diminui. Entretanto, a incidência na reabertura de bugs aumenta, ou seja, a quantidade de vezes que o mesmo bug é reaberto aumenta. Isso pode ser justificado por causa da expertise dos desenvolvedores [23], que aumenta conforme o tempo avança, pode ser também justificado pelo uso de ferramentas mais elaboradas para a correção dos bugs, fazendo com que a reabertura de bugs diminua com o passar dos anos.

4.2 Execução do Experimento

Após a execução do experimento, obtivemos os resultados apresentados na Tabela 6 e na Figura 5.

Modelo/Métrica	Acurácia	Precisão	Cobertura	F1
Naive Bayes	58,83%	88,62%	14,92%	25,55%
Decision Tree	75,12%	75,39%	70,42%	72,82%
k-Nearest Neighbor (KNN)	75,45%	75,63%	74,14%	74,87%

Tabela 6: Métricas dos modelos.

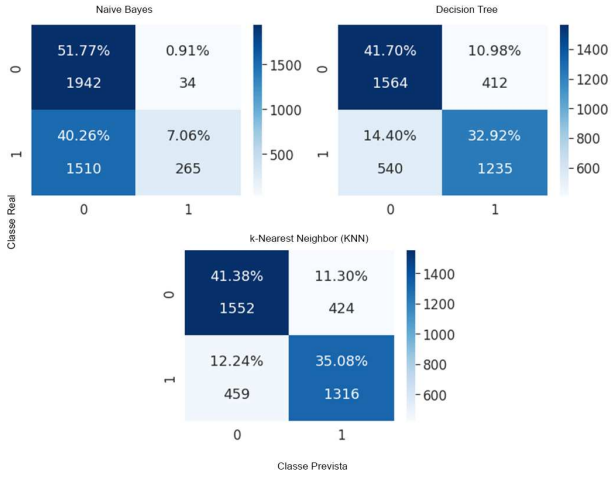

Figura 5: Matrizes de confusão dos modelos.

Como podemos observar, o modelo KNN foi o que obteve melhores resultados, tanto predizendo assertivamente os bugs reabertos quanto os bugs não reabertos. Esse modelo traz os melhores resultados para acurácia, cobertura e F1, e o segundo melhor para precisão. Esse resultado indica que, dentro do *dataset*, o que pode definir para um bug ser reaberto está relacionado a mais de uma *feature*, fazendo com que seja necessário analisar a relação de cada *feature* com o status do bug e identificar quais *features* fazem com que o bug seja reaberto.

Também podemos visualizar que o modelo *Decision Tree* também obteve bons resultados, semelhantes ao modelo KNN, com ótimos valores para as métricas. Isso indica que, semelhante ao KNN, a ocorrência de um bug ser reaberto está relacionada a mais de uma *feature*.

Já o modelo de Naive Bayes, obteve valores bem baixos de cobertura e f1, dado que a porcentagem de classificação de bugs reabertos como bugs não reabertos foi muito elevada, enquanto a porcentagem de classificação de bugs reabertos como bugs reabertos foi muito baixa, fazendo assim com que essas métricas não obtivessem bons resultados. Como o modelo Naive Bayes define que todas as *features* possuem a mesma importância para a classificação, esse modelo não obteve bons resultados, também indicando que algumas *features* possuem mais relevância do que outras, no que se diz respeito à reabertura do bug. Podemos ver mais detalhadamente as observações descritas entre os modelos na Figura 6.

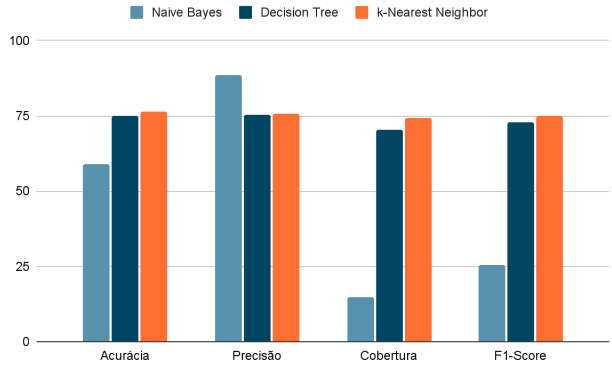

Figura 6: Gráfico de comparação dos modelos.

4.2.1 Respondendo à Questão de Pesquisa 1

Tentando responder à QP1, realizamos uma análise mais refinada e mais precisa, também efetuando um estudo focado em cada *feature* do *dataset*, tornando mais fácil a obtenção das *features* que podem influenciar a um bug ser reaberto. Para tanto, iniciamos com a construção de um gráfico de correlação, como podemos ver na Figura 7.

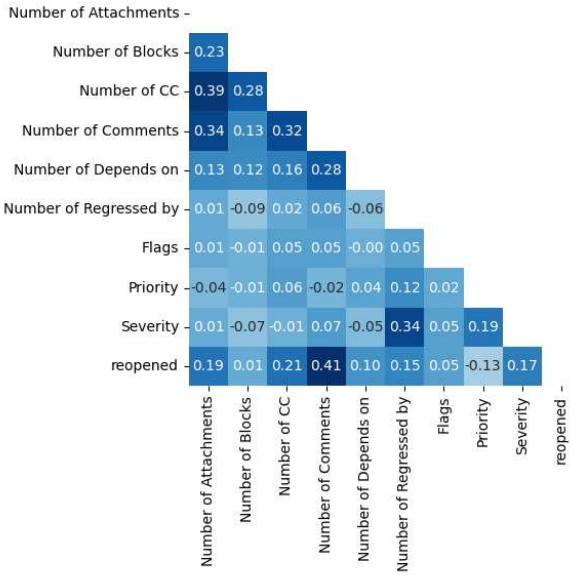

Figura 7: Gráfico de correlação entre as *features*

Como podemos observar, a *feature Number of Comments* é a que possui mais correlação com a *feature reopened*, seguida de *Number of CC, Number of Attachments, Severity, Number of Regressed by, Number of Depends on, Flags, Number of Blocks* e *Priority*. Esses valores indicam a relação de cada *feature* com a reabertura do bug. Assim, a *feature Number of Comments* é a que possui maior relação.

Após a análise da correlação das *features* com a reabertura do bug, executamos o experimento com as *features* isoladas e com o modelo que obteve melhores resultados no

experimento anterior, KNN, obtendo os resultados ilustrados na Tabela 7.

Feature	Acurácia	Precisão	Cobertura	F1
Number of Comments	60,83%	60,28%	50,53%	54,98%
Number of CC	56,62%	54,45%	50,98%	52,66%
Number of Attachments	45,40%	46,08%	89,23%	60,78%
Severity	55,29%	100%	5,52%	10,46%
Number of Regressed by	55,18%	98,95%	5,35%	10,15%
Number of Depends on	46,89%	47,00%	95,77%	63,05%
Flags	47,32%	47,32%	100%	64,24%
Number of Blocks	48,57%	46,66%	60,67%	52,75%
Priority	57,79%	53,70%	78,42%	63,75%

Tabela 7: Métricas usando KNN para cada *feature* isolado.

Esses resultados evidenciam que uma única *feature* não define um bug a ser reaberto. Todos os resultados de acurácia ficaram abaixo dos 61%, ou seja, a performance de classificação dos bugs, quando utilizado as *features* em separado, não foi tão boa. Entretanto, a maioria das *features*, com exceção de *Number of Attachments*, *Number of Depends on*, *Flags* e *Number of Blocks*, tiveram acurácia acima dos 55%, evidenciando que elas são relevantes para a classificação dos bugs. Quando executamos o experimento, usando KNN, sem levar em consideração essas 4 *features*, obtemos os resultados vistos na Tabela 8.

Acurácia	Precisão	Cobertura	F1-Score
70,86%	70,94%	65,07%	67,88%

Tabela 8: Métricas usando KNN usando apenas as *features* que obtiveram acurácia acima de 55%.

Observando os resultados da Tabela 10 e comparando com os resultados da Tabela 8, para o modelo KNN, vimos que os resultados foram bem próximos, variando em aproximadamente 5%, para acurácia e precisando, e aproximadamente 9%, para cobertura e F1-Score, indicando que as *features* que obtiveram acurácia acima de 55% estão fortemente relacionadas com a reabertura do bug. Assim, tentando responder à QP1 da nossa pesquisa, observamos que a reabertura de um bug se dá através de um conjunto de *features*, mostrando que as *features* em separado não possuem tanta influência para tal.

5. AMEAÇAS À VALIDADE

Nesta seção, são apresentadas algumas ameaças à validade desta pesquisa e a maneira como foram amenizadas.

Apesar da utilização de ferramentas automáticas para a busca de dados e classificação, grande parte do processo de agrupamento e preparação dos dados foi feito por análise humana, impondo um erro inerente à pesquisa. Algumas ameaças à validade desta pesquisa podem estar relacionados à construção do *dataset*, podendo haver viés no ato de escolhas de *features* utilizadas para classificação. Por isso, foi utilizado um processo de definir critérios de qualidade, para garantir um resultado confiável.

Neste trabalho, foi assumido que as ferramentas de busca de bugs utilizada, Bugzilla, funciona corretamente e reflete uma gama de projetos diversificados, refletindo a realidade. Por isso, não se utilizou filtros para captura de bugs de projetos específicos, a fim de englobar bugs de diferentes projetos.

6. TRABALHOS RELACIONADOS

ZIMMERMANN [2], em seu trabalho, utilizou um survey para estudar e tentar entender o processo de reabertura de bugs. Usando os resultados obtidos através das respostas do survey, foi possível identificar que bugs abertos pelos clientes ou encontrados durante a execução de testes do sistema são mais propensos a serem reabertos, devido à dificuldade e complexidade de reproduzir e pela dificuldade da escrita dos relatórios de bugs, pois, como são clientes, não possuem elevado grau de destreza para essa tarefa. Também foi indicado que bugs assinados com severidade inicial mais alta estão mais propensos a serem reabertos. Assim, podemos perceber que há relação entre a severidade do bug e a reabertura do mesmo.

No trabalho de SOUZA e CHAVEZ [22], foi realizado um estudo sobre a reabertura de bugs, trazendo impactos e prevenção. Nele, são descritas algumas causas de reaberturas de bugs em alguns projetos de software livre. Essas descrições indicam que alguns bugs foram reabertos porque a correção gerou novos defeitos, o que podemos identificar que a *feature "Number of Regressed by"* possui relevância para a reabertura do bug. Outra causa encontrada foi que bugs foram reabertos porque não havia informação suficiente para a compreensão do defeito, e, em outros casos, foi marcado como resolvido por engano. Assim, sendo possível traçar relações entre as *features* estudados na nossa pesquisa com as descrições feitas no trabalho analisado.

No trabalho de MI e KEUNG [23], foram apresentadas algumas informações importantes e que se relacionam diretamente com nosso trabalho. Nela, é descrito que bugs reabertos tendem a ter mais comentários, dado o seu ciclo de vida maior e as discussões que geram nesses bugs. Além disso, a pesquisa também aponta causas de reabertura de bugs relacionadas à descrição não clara do bug, negligência do desenvolvedor ao tentar corrigir o bug, criação de outros bugs através da correção do bug original ou até porque, depois, foi encontrada uma solução melhor para a correção de um bug que já foi corrigido.

O nosso trabalho se diferencia em algumas questões, quando comparado com o trabalho de ZIMMERMANN [2], SOUZA e CHAVEZ [22] e MI e KEUNG [23]. Nosso trabalho

não utiliza survey para o estudo, faz uso de um dataset de bugs para treinar algoritmos de classificação. Além disso, no nosso trabalho, utilizamos *features* a mais do que nesses trabalhos apresentados.

Sendo assim, através desses trabalhos, podemos nos nortear melhor em como refinar as buscas de *features* candidatas e desenvolver melhorias no experimento executado, além de uma melhor compreensão sobre bugs reabertos.

7. CONCLUSÕES

Através da execução do experimento de análise de *features* que possuem relação com a reabertura do bug, e análise dos resultados, foi possível constatar que há *features* que influenciam mais na reabertura de bugs, são elas: *Number of Comments, Number of CC, Severity, Number of Regressed by* e *Priority*. No entanto, também foi possível visualizar que as *features* em separado não possuem uma grande influência de reabertura de bugs, ou seja, uma única *feature* não tem poder suficiente para definir que um bug será reaberto. Além disso, foram estudados três algoritmos de classificação (*Naive Bayes, Decision Tree* e KNN) para classificar os bugs e, com isso, foi identificado que o algoritmo KNN alcança melhores resultados para esse tipo de classificação.

Também foi possível identificar quais são as *features* que possuem relação com o evento de reabertura do bug, são elas: número de comentários, número de pessoas interessadas, severidade, número de bugs que foram inseridos após a correção do bug e prioridade.

Com essas conclusões, podemos identificar que a escrita do relatório de bugs pode ser melhorada, adicionando mais detalhes do bug, definindo níveis de severidade e prioridade adequados ao problema. Além disso, no processo de correção do bug, é importante a validação da solução ser mais adequada e mais eficiente, evitando a abertura de novos bugs e a reabertura de bugs antigos.

Outra consideração importante é a validade da construção do *dataset*, até que ponto, as *features* selecionadas tendem a contribuir para a reabertura do bug ou se elas apenas crescem devido ao tempo de vida que um bug tem ou outro tipo de influência que não seja diretamente a reabertura do bug. Por exemplo, algumas dessas *features* podem ser enviesadas pelo tempo que o bug ficou aberto, assim, pensando logicamente, o número de comentários poderia tender a aumentar conforme a quantidade de tempo que o bug fica aberto.

Para trabalhos futuros, pode-se partir da análise mais minuciosa sobre cada *feature*, a fim de reduzir apenas às *features* que realmente causam a reabertura do bug. Também, pode-se levar em consideração quais agrupamentos de *features* possuem mais relevância para a classificação de bugs como reabertos. Assim, este trabalho pode ser utilizado em outras pesquisas relacionadas ao estudo de reabertura de bugs no Bugzilla, fazendo com que processos iniciais e tarefas custosas sejam puladas.

8. AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus, pelo dom da vida e por ter me ajudado e me acompanhando durante a execução deste trabalho, me dando forças para prosseguir, resiliência para não desistir e sabedoria para decidir.

Também agradeço aos meus pais, Euclides Araújo e Ogla Araújo, pelos ensinamentos, desde cedo, e por me dar total apoio em todas as minhas decisões. Sem dúvidas, vocês são os meus maiores exemplos de vida e meus pilares. Agradeço também à minha noiva, Mirella Trajano, que esteve sempre ao meu lado durante toda essa trajetória, sendo meu apoio, minha confidente e disponível para tudo mais que eu pudesse precisar. Agradeço à minha irmã, Yasnaya Delly, por todo o suporte dado e por sempre estar presente nos momentos que eu mais precisei, pelas palavras de incentivo e pelos risos proporcionados.

Agradeço também ao meu orientador, Franklin Ramalho, que me deu muita ajuda e direcionamento na execução deste trabalho. Sou grato também por ter os melhores amigos que eu poderia imaginar. Edson Weslley, Lucas Abrantes e Igor Filho, obrigado por toda a parceria e amizade durante todo o curso, por todos os momentos bons, de descontração, e por sempre estarem disponíveis para me ajudar. Também agradeço aos meus amigos, Brenda e Gustavo, sempre trazendo leveza ao ambiente, me incentivaram a não desistir e a perseverar até o fim. Vocês, meus amigos, romperam os muros da universidade e se tornaram amigos que levarei para toda a vida.

Obrigado a todos vocês que fazem parte da minha história e contribuíram para que eu pudesse chegar até aqui.

9. REFERÊNCIAS BIBLIOGRÁFICAS

[1] BAUMGARTNER, Cristiano. Conheça a importância de identificar bugs preventivamente no desenvolvimento de software. Testing Company, 2021. Disponível em: https://testingcompany.com.br/blog/conheca-a-importancia-de-identificar-bugs-preventivamente-no-desenvolvimento-de-software. Acesso em: 15 de março de 2022.

[2] ZIMMERMANN, Thomas. et al. Characterizing and Predicting Which Bugs Get Reopened. International Conference on Software Engineering (ICSE), Zurich, Switzerland, 34th, Páginas 1074-1083, 2012.

[3] ALMOSSAWI, Ali. Investigating the architectural drivers of defects in open-source software systems: an empirical study of defects and reopened defects in GNOME. 2012. Tese (Mestrado) - Curso de Ciências em Engenharia e Gestão, Instituto de Tecnologia de Massachusetts, Massachusetts, 2012.

[4] COSSETTI, Melissa. O que sabemos sobre o ataque ao Facebook que afetou 90 milhões de contas. Tecnoblog, 2019. Disponível em: https://tecnoblog.net/especiais/melissa-cruz-cossetti/o-que-sabemos-sobre-o-ataque-ao-facebook/. Acesso em: 25 de março de 2022.

[5] BUGZILLA. About Bugzilla, Bugzilla, 2021. Disponível em: https://www.bugzilla.org/about/. Acesso em: 23 de março de 2022.

[6] BUGZILLA, 2022. User Guide, Bugzilla, 2022. Disponível em:

https://wiki.mozilla.org/BMO/UserGuide/BugFields#assigned_to. Acesso em: 23 de março de 2022.

[7] FACELI, K.; LORENA, A. C.; GAMA, J. ; CARVALHO, A. C. P. L. F. de. Inteligência artificial: uma abordagem de aprendizado de máquina. Rio de Janeiro: LTC, 2011.

[8] SILVA, Adilane Ribeiro. Uma visão geral sobre Machine Learning, 2020. Disponível em: https://operdata.com.br/blog/uma-visao-geral-sobre-machine-learning/. Acesso em 06 de agosto de 2022.

[9] RUDINEY. Modelos de Classificação: aprendizado de máquina para iniciantes, 2021. Disponível em: https://www.digitalhouse.com/br/blog/modelos-de-classificacao/. Acesso em 06 de agosto de 2022.

[10] REMIGIO, Matheus. Modelo Bayesiano - Naive Bayes, 2020. Disponível em: https://medium.com/@msremigio/modelo-bayesiano-naive-bayes-f4b1f4d61424. Acesso em 12 de agosto de 2022.

[11] REMIGIO, Matheus. Árvores de Decisão - Decision Tree, 2020. Disponível em: https://medium.com/@msremigio/%C3%A1rvores-de-decis%C3%A3o-decision-trees-4cb6857671b3. Acesso em 12 de agosto de 2022.

[12] REMIGIO, Matheus. Aprendizagem Baseada em Instâncias - KNN, 2020. Disponível em https://medium.com/@msremigio/aprendizagem-baseada-em-inst%C3%A2ncias-knn-7e2c6f0778bc. Acesso em 12 de agosto de 2022.

[13] BECKER, Lauro. Algoritmo de Classificação Naive Bayes, 2019. Disponível em: https://www.organicadigital.com/blog/algoritmo-de-classificacao-naive-bayes/. Acesso em 06 de agosto de 2022.

[14] GARCIA, Simone Carboni. O Uso de Árvores de Decisão na descoberta de conhecimento na área da saúde, 2003. Tese (Mestrado) - Curso de Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

[15] SACRAMENTO, Gabriel. Árvore de decisão: entenda esse algoritmo de Machine Learning, 2021. Disponível em: https://blog.somostera.com/data-science/arvores-de-decisao. Acesso em: 06 de agosto de 2022.

[16] FERRERO, Carlos Andres. Algoritmo KNN para previsão de dados temporais: funções de previsão e critérios de seleção de vizinhos próximos aplicados a variáveis ambientais em limnologia, 2009. Tese (Mestrado) - Curso de Ciências de Computação e Matemática Computacional, Universidade de São Paulo, São Carlos, 2009.

[17] COELHO, Paulo S. S. LACHTERMACHER, Gerson. EBECKEN, Nelson. Classificação de Dados: uma visão geral, 2003. Encontro da Associação Nacional de Pós-Graduação e Pesquisa em Administração, 2003.

[18] SCUDILIO, Juliana. Qual a melhor métrica para avaliar modelos de Machine Learn?, 2020. Disponível em: https://www.flai.com.br/juscudilio/qual-a-melhor-metrica-para-avaliar-os-modelos-de-machine-learning/. Acesso em: 06 de agosto de 2022.

[19] FIREFOX. Priority Definitions. Disponível em: https://firefox-source-docs.mozilla.org/bug-mgmt/guides/priority.html. Acesso em: 06 de agosto de 2022.

[20] BUGZILLA. BOM/UserGuide/BugFields, 2022. Disponível em: https://wiki.mozilla.org/BMO/UserGuide/BugFields#bug_severity. Acesso em: 06 de agosto de 2022.

[21] SCIKIT-LEARN. Supervised Learning. Disponível em: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning. Acesso em 13 de agosto de 2022.

[22] SOUZA, Rodrigo; CHAVEZ, Christina. Characterizing Verification of Bug Fixes in Two Open Source IDEs, 2012. In: 9th Working Conference on Mining Software Repositories (MSR 2012), 2012, Zurich. Proc. of the 9th Working Conference on Mining Software Repositories, 2012. p. 70-73.

[23] MI, Qing. KEUNG, Jacky. An empirical analysis of reopened bugs based on open source projects, 2016. In Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE '16). Association for Computing Machinery, New York, NY, USA, Article 37, 1–10.

[24] BUGZILLA. 4.4. Classifications, Products, Components, Versions, and Milestones. Disponível em: https://bugzilla.readthedocs.io/en/latest/administering/categorization.html. Acesso em: 19 de agosto de 2022.