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

ÁXEL CRISPIM E MEDEIROS

INVESTIGANDO ABORDAGENS PARA MELHORIAS NAS ESCRITA DE RELATÓRIOS DE BUGS

CAMPINA GRANDE - PB

2021

ÁXEL CRISPIM E MEDEIROS

INVESTIGANDO ABORDAGENS PARA MELHORIAS NAS ESCRITA DE RELATÓRIOS DE BUGS

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: Professor Dr. Franklin de Souza Ramalho.

CAMPINA GRANDE - PB

2021

ÁXEL CRISPIM E MEDEIROS

INVESTIGANDO ABORDAGENS PARA MELHORIAS NAS ESCRITA DE RELATÓRIOS DE BUGS

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:

Professor Dr. Franklin de Souza Ramalho

Orientador – UASC/CEEI/UFCG

Professora Dr. Everton Leandro Galdino Alves

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 30 de Março de 2022.

CAMPINA GRANDE - PB

RESUMO

No desenvolvimento de software, os relatórios de bugs variam em qualidade, completude e precisão nas suas descrições. Normalmente, os relatos mal redigidos são elaborados por usuários sem qualquer instrução técnica, o que requer mais tempo para compressão do bug, por meio de discussões entre os desenvolvedores e relatores. Entretanto, apesar de existirem plataformas com ferramentas para auxiliar a publicação dos relatórios, como o Bugzilla, ainda é frequente a necessidade de discussões, o que faz necessário mais estudos para entender a raiz deste problema. Portanto, este trabalho propõe uma investigação nos campos de descrição e nas interações dos usuários em relatórios de bugs, feitos na plataforma Bugzilla, com objetivo de identificar padrões nos posts que influenciam no período de tempo para solução ou descarte destes. Nesse contexto, a pesquisa será conduzida por estudo quantitativo com Machine Learning(ML) que busca relacionar o tempo de resolução, abertura até o fechamento, com uma estruturação dos campos pertencentes aos relatórios, como: descrições, comentários, prioridade, se foi confirmado ou não, número de comentários, etc. Como resultado, encontrou o intervalo de 8 à 25 comentários como ideal para resolução dos relatórios e houve uma limitação com o trabalho com o processo de estruturação, assim, para os trabalhos futuros espera-se refazer o estudo usando uma abordagem qualitativa ou utilizar uma ferramenta externa para a estruturação.

Investigando Abordagens para Melhorias nas Escrita de Relatórios de Bugs

Áxel Crispim e Medeiros
axel.medeiros@ccc.ufcg.edu.br
Universidade Federal de Campina Grande Campina Grande, Paraíba

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

RESUMO

No desenvolvimento de software, os relatórios de *bugs* variam em qualidade, completude e precisão nas suas descrições. Normalmente, os relatos mal redigidos são elaborados por usuários sem qualquer instrução técnica, o que requer mais tempo para compressão do *bug*, por meio de discussões entre os desenvolvedores e relatores. Entretanto, apesar de existirem plataformas com ferramentas para auxiliar a publicação dos relatórios, como o Bugzilla, ainda é frequente a necessidade de discussões, o que faz necessário mais estudos para entender a raiz deste problema. Portanto, este trabalho propõe uma investigação nos campos de descrição e nas interações dos usuários em relatórios de bugs, feitos na plataforma Bugzilla, com objetivo de identificar padrões nos posts que influenciam no período de tempo para solução ou descarte destes. Nesse contexto, a pesquisa será conduzida por estudo quantitativo com *Machine Learning*(ML) que busca relacionar o tempo de resolução, abertura até o fechamento, com uma estruturação dos campos pertencentes aos relatórios, como: descrições, comentários, prioridade, se foi confirmado ou não, número de comentários, etc. Como resultado, encontrou o intervalo de 8 à 25 comentários como ideal para resolução dos relatórios e houve uma limitação com o trabalho com o processo de estruturação, assim, para os trabalhos futuros espera-se refazer o estudo usando uma abordagem qualitativa ou utilizar uma ferramenta externa para a estruturação.

PALAVRAS-CHAVE

Bug reports, Bugzilla, Machine learning, Engenharia de software

1. INTRODUÇÃO

Os relatórios de *bugs* são solicitações, regularmente com origem na sua comunidade de usuários, que descrevem mudanças a serem analisadas e aplicadas no software [1]. Eles consistem em duas categorias: descrição de *bugs* e requisição de *features*. A primeira possui o propósito de incentivar a correção de erros ou falhas, enquanto que a segunda visa descrever uma adição de uma nova funcionalidade.

De maneira geral, um relatório é formado por uma descrição em linguagem natural [2], e é decomposto em vários campos, como, por exemplo: informações de plataforma, passos de reprodução, resultados esperados/atuais e casos de teste. E, por ser escrito em linguagem natural, a sua falta de estrutura causa ambiguidades e imprecisões que, aliadas com desvios de organização lógica, podem ter um impacto negativo na sua compreensão.

Portanto, a construção de um relatório de *bug* não é uma atividade trivial, devido ao seu formato, conhecimento prévio/contexto, ou a simples distinção entre erro de configuração ou *bug* não serem habituais aos usuários [3]. Logo, torna-se uma barreira para os usuários que reportem os *bugs*, como também uma dificuldade na comunicação com os desenvolvedores, ao lerem relatos mal formulados, incompletos ou confusos. Nesse contexto, os desenvolvedores enfrentam vários problemas ao analisar relatórios de *bugs*, como impossibilidade de sua reprodução. Por via de regra, isso acontece devido a informações ausentes e imprecisas, até mesmo incorretas nas descrições dos relatórios [4]. Além disso, existem algumas informações importantes que os usuários não conhecem, como *Stack Traces*[1] e dados de rastreamento de *bugs*, cuja obtenção é feita por consulta manual em logs da aplicação ou outro recurso de software para análise de erros, como por exemplo, *debuggers*[2] .

Para contornar estes problemas, plataformas de publicação dos relatórios, como o Bugzilla [5], têm equipes de suporte que buscam orientar usuários por meio de um sistema de perguntas e respostas, estilo um fórum na internet. Assim, após o usuário publicar um relato, o suporte irá realizar comentários para elucidar e adquirir informações relevantes para o contexto do *bug* como, por exemplo, passos para reproduzir, qual é o resultado esperado e o atual, quais são casos de testes, sua frequência, entre outros. Dessa maneira, esse processo depende da disponibilidade de tempo e da comunicação entre os usuários, no qual, caso esses recursos não estejam disponíveis, ocorre um aumento no tempo necessário para solução do relato. Portanto, por meio de uma descrição robusta do *bug*, a equipe de desenvolvimento pode focar na solução a priori e economizar tempo, recurso importante em projetos de software, que é gasto na orientação dos usuários.

A importância dos relatórios de *bug* reside em seu papel na evolução do software, pois orientam futuras atualizações no software de acordo com os anseios dos usuários, tornando o software mais eficaz para a comunidade e com uma maior usabilidade. Adicionalmente, eles são fundamentais para prover uma maior estabilidade, ou seja, garantir que o software tenha o mínimo de erros possível, conforme o efeito da Lei de Linus [6].

---

[1] *Stack Traces:* Conhecidos como rastreamento de pilha, são informações para debugger que exibem a pilha de execução de um programa ou ordem da chamada funções executadas. Muito utilizado para situações quando o programa é interrompido na sua execução.

[2] *Debuggers*: Ferramentas fornecidas por linguagens ou ambientes de programação que permitem acompanhar a mudança de estado das variáveis dos programas em nível de execução.

Em síntese, os relatórios de *bugs* melhoram a qualidade do software.

A proposta deste artigo é investigar as relações entre os campos dos relatórios e as interações entre os usuários na plataforma Bugzilla [5], com o intuito de determinar como estes fatores influenciam o tempo de resolução do *bug* descrito no relatório. Além disso, buscamos entender qual a relação desses fatores com relatórios ainda abertos, com o objetivo de indicar possíveis causas para sua não conclusão. E para esta abordagem, utilizamos a aplicação de algoritmos de aprendizagem supervisionada, como *RandomForest* [7], Árvore de Decisão [8] e KNN [9], para predizer a variável tempo de resolução por meio de uma classificação.

O documento está organizado como segue. A seção 2 apresenta a fundamentação teórica justificando e explicando conceitos importantes para o melhor entendimento deste trabalho. A seção 3 apresenta os trabalhos relacionados, enquanto a seção 4 explica a técnica aplicada. A seção 5 explica como foi realizado o experimento, enquanto a seção 6 apresenta e discute os resultados do experimento. Por fim, a seção 7 apresenta as considerações finais sobre o trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

O termo aprendizado de máquina refere-se a modelos computacionais com a capacidade de reconhecer padrões automaticamente [10], e como tal, é frequentemente aplicado a bancos de dados complexos o suficiente para superar as limitações do entendimento humano, o que impossibilita a sua programação por divisão de instruções, como o método tradicional de escrever algoritmos. E por identificar padrões, as soluções de ML são genéricas e aplicáveis a diferentes contextos, como pesquisa científica, otimização do uso de energia, controle de tráfego de rede, etc. [10]

Outro fator importante, é que algoritmos de ML são adaptáveis à evolução, pois sua solução é capaz de garantir o desempenho à medida que há mudanças nos dados fornecidos. Isso se deve por seu aprendizado utilizar funções de avaliação de desempenho [11] que retroalimentam o modelo e aprimoram os seus resultados, logo, aprimorando a experiência do modelo com o passar do tempo.

De acordo com os processos de aprendizagem, abordagens de ML podem ser categorizadas em: aprendizagem supervisionada e não supervisionada [12]. No primeiro, os padrões são identificados fornecendo um conjunto de rótulos que mapeiam pares de entradas-saída, como utilizado neste estudo. No segundo, não há nenhum rótulo fornecido, assim, torna-se função dos algoritmos utilizar estratégias para identificação dos padrões, como: aprendizagem por reforço, estimação de densidade, formação de agrupamentos, dentre outros [13].

Como ML trabalha reconhecendo padrões, uma de suas aplicações é a classificação de valores categóricos, o que auxilia na análise de valores com maior poder descritivo, como “Até 18 anos” para classificação das idades de menores. Diante disso, selecionamos um subconjunto de algoritmos populares na literatura, para realizar a aprendizagem supervisionada neste estudo, assim, temos:

- Árvore de Decisão: funciona selecionando os parâmetros capazes de dividir o *dataset* em um menor número de partições, realizando esse processo recursivamente até que a árvore gerada consiga expressar as regras de construção do *dataset.*
- *RandomFlorest*: É uma técnica que gera um finito número de árvores de decisões utilizando *features* aleatórias e, ao final, ponderando sobre a média dos resultados para encontrar uma solução mais eficaz para predição do *dataset.*
- KNN: o algoritmo que busca determinar o valor de uma amostra calculando a menor distância entre os K-vizinhos, por isto, os K-vizinhos são definidos como os valores valores mais próximos do conjunto de dados. Assim, a classificação é realizada com uma contagem de frequência entre os rótulos dos seus K-vizinhos para encontrar o rótulo final da amostra.

3. TRABALHOS RELACIONADOS

O principal problema com plataformas abertas de *bugs* é o custo necessário para solução do *bug* [14], devido às diversas variáveis como a quantidade de *bugs* abertos, quantidades de desenvolvedores, conhecimento específico de cada desenvolvedor, tempo disponível, entre outros. Assim, a literatura busca várias formas para auxiliar neste custo, como Anvik [14] que propôs uma ferramenta semi automatizada para entrega de relatório para o desenvolvedor com maior probabilidade de solucionar o relato, utilizando ML para enriquecer a inteligência do processo. Diferentemente, nosso trabalho busca compreender as influências no tempo de resolução do relato com o auxílio de ML.

Para contribuir na questão, outras pesquisas visam compreender o que torna um bom relatório de *bug*, que informações contém, sua frequência e impacto dessas informações do ponto de vista dos desenvolvedores, como realizado por Battenburg [4]. O pesquisador concluiu que o resultado observado e esperado é encontrado em mais de 50% dos *reports*, entretanto, *stack traces* e casos de teste são quase inexistentes, apontados como informações muito relevantes pelos desenvolvedores.

Já em 2010, Davies [3] realizou um estudo para descobrir a influência da composição da descrição do relato no processo de correção de *bugs*. Para isso, utilizou *features* para padronizar e comparar relatos com diferentes origens e níveis de completude, como: *Expected behaviour, Steps to reproduce, Test cases, Build information*, etc. De forma semelhante, utilizamos essas *features* para a padronização da escrita dos relatos, apenas adaptando essa estrutura com a adição de dados da plataforma (*User Agent, Platform*) e configuração iniciais (*Precondition*), além de remover o casos de testes por não encontrar essa padrão na escrita dos relatórios da plataforma Bugzilla.

E conforme defendido por Battenburg [2], informações estruturadas possuem uma melhor performance do que texto livre em abordagens com *Machine Learning*. Logo, o pesquisador propôs uma ferramenta que implementa a estruturação automática em *bug reports*, chamada de Infozilla[3], com a conclusão que o resultado encontrado era satisfatório para dados com diferentes níveis de granularidade na escrita do relato de *bug*.

Com base no estudo feito por Rodrigues [15], que busca realizar a predição do tempo de permanência dos estudos do ensino superior brasileiro, os algoritmos *RandomForest*, Árvore de Decisão,

[3] https://github.com/kuyio/infozilla.

XGBoost indicaram bons resultados ao realizar a predição de variáveis categóricas referentes ao tempo, mesmo parâmetro de avaliação utilizado em nosso trabalho. Portanto, utilizamos o mesmo processo de predição de variáveis categóricas para classificação do tempo, com o nosso trabalho se diferenciando com o contexto de relatos de *bug* e a substituição de *XGBoost* pelo algoritmo KNN.

4. TÉCNICA PROPOSTA

Este trabalho é um estudo quantitativo, e exploratório, realizado em uma base de relatos da plataforma Bugzilla, com objetivo de compreender as relações entre campos e interações entre usuários em relatórios de *bugs,* utilizando como parâmetro o tempo de resolução dos *bug reports*. Nesse caso, o tempo de resolução representa todo o período de abertura do relato, calculado com a diferença entre a data da postagem e de conclusão[4]/última modificação. Com base nisso, os relatos são classificados em dois tipos: em execução (abertos) e concluídos (fechados). Assim, de forma específica, este estudo visa responder às seguintes Questões de Pesquisa: (QP1) Quais são os principais parâmetros (*features*) que influenciam no tempo de resolução do relatório de *bugs* fechados? (QP2) Qual é a principal diferença entre relatórios de bugs abertos e fechados?

Para responder às questões acima mencionadas, foi realizada uma análise na API do Bugzilla [16] e foram selecionadas *features* que indicam status do relato como identificadores de tempo (início, atribuído e conclusão) do relato, se foi solucionado ou não, severidade do *report*, contagem de comentários, classificação da origem do relato (cliente, servidor, infraestrutura, etc) e a prioridade. Além disso, observamos a necessidade de *features* que indiquem a estrutura da escrita da postagem, porém, não existe esta estrutura na API da plataforma. Assim, de forma experimental, utilizamos expressão regular para construção de um script para extrair os seguintes campos::

- Passos de reprodução: descreve a sequência de passos a serem realizados para a reprodução do *bug*;
- Precondições: toda e qualquer configuração especial que deve ser realizada antes da execução dos passos de reprodução, como configuração de ambiente, plugins, etc;
- Resultado Atual: resultado que o usuário está recebendo atualmente, que representa uma situação de erro ou problema;
- Resultado Esperado: resultado que o usuário esperava ter ao executar uma determinada atividade;
- Reprodutibilidade: frequência de reprodução de um relato, como: sempre, às vezes, de vez em quando, etc. Muito utilizada para entender se o problema se trata de uma execução síncrona ou assíncrona;
- Plataforma: são informações de *build,* do sistema operacional ou alguma configuração de ferramenta presente.

Para realizar a análise, definimos "*time_class*", uma variável correspondente à classificação do tempo de resolução, em dias, para categorizar as seguintes valores de passagens no tempo: no mesmo dia (DX_1), entre 2 à 10 dias (D2_10), entre 11 e 30 dias (D11_30), entre 31 à 60 dias (D31_60) e com mais de 60 dias (D61_Y). Dessa maneira, buscamos categorizar todas as possíveis passagens de tempo da plataforma e representar o tempo de resolução em variáveis mais descritivas, para facilitar a discussão a posteriori no documento.

Por fim, usamos a biblioteca MARS [17] para selecionar as melhores *features* candidatas [18], que são o menor subconjunto de características capazes de produzir classificações comparáveis a modelos mais complexos.

[4] Marcado com o rótulo "*Closed*" ao lado do título na plataforma Bugzilla.

5. O EXPERIMENTO

5.1 Features

A plataforma Bugzilla fornece 03 (três) *endpoints* para acesso dos dados dos *bug reports*, são eles: *summary, comment* e *history*. O summary resume as principais informações, como estado de resolução, identificadores (*id*), se foi confirmado e dependências com outros relatos. Já *comment* e *history* possuem os dados brutos dos comentários e as mudanças de metadados do fórum, respectivamente. As *features* utilizadas no estudo são explicadas na Tabela 1

Origem	Nome	Descrição
API Bugzilla	id	Identificador do bug
	priority	Prioridade da plataforma ("--" e P1 até P5), com a prioridade diminuindo de acordo com o aumento do número, ou seja, P1 > P2 > …P5 > "--" [19].
	severity	Severidade do relato: major, critical, normal, minor, trivial, enchantment, N/A, S1, S2, S3, S4, '--'.
	status	Representa o nível de verificação do relato pela equipe de suporte, classificado em: UNCONFIRMED, NEW, ASSIGNED, REOPENED, RESOLVED, VERIFIED e CLOSED.
	resolution	É o estado de resolução ou correção do relato, dividido em: NEW, FIXED,, INVALID,, WONTFIX,, INACTIVE,, DUPLICATE, WORKSFORME.
	type	Definição do tipo de relato, sendo: defect, enchantment. Usado relatos do tipo defeito no estudo.

	comments	É o texto da postagem do relato
	comment_count	Contagem de comentários da postagem incluindo a postagem inicial
	1.initialTime 2.assignedTime 3.closedTime	Datas da postagem, assinatura da equipe de suporte e fechamento do relato, respectivamente.
Dados Estruturados	1.Plataform_text 2.plataform_size 3.plataform_numberLines 4.UserAgent_text 5.UserAgent_size 6.UserAgent_numberLines	Dados da plataforma do bug.
	1.StepsToRepoduce_text 2.StepsToRepoduce_size 3.StepsToRepoduce_numLines 4.Precondition_text 5.Precondition_numberLines 6.Precondiiton_size	Referentes aos passos para reprodução e configurações de ambiente
	1.ExpectedResult_text 2.ExpectedResult_size 3.ExpectedResult_numberLines 4.ActualResult_size 5.ActualResult_text 6.ActualResult_numberLines	Referentes aos resultados esperados e os os atuais ao executar uma determinada atividade.

Tabela 1: *Features* selecionadas para o experimento.

Para realizar a extração da postagem, foi observado um padrão na escrita dos relatos que diferenciava o conteúdo da postagem em tópicos, semelhante aos campos utilizados por Davies [3]. Logo, utilizamos expressões regulares para extrair o conteúdo das tags (*Prerequisites*, *Steps to reproduce*, etc.) usando um filtro para padronizar as diferentes formas de escrita, como "*STR*", "*Steps*", "*Steps to reproduce*", etc. Mostramos, na Figura 1, um exemplo de uma postagem com o padrão detalhado acima.

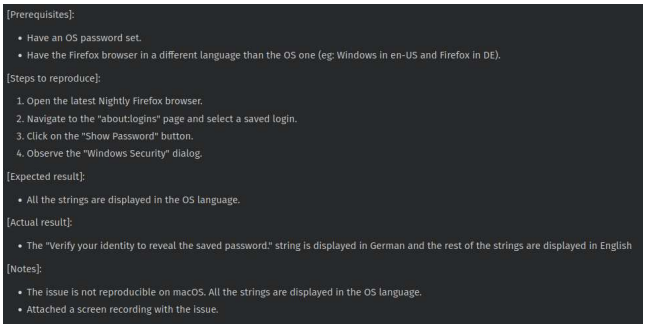

Figura 1: Bug 1629538 do Bugzilla [20].

Na Figura 1, observamos que existe um padrão de separação entre as *tags* e os seus conteúdos, de acordo com: [*tag* 1], conteúdo da *tag* 1, [*tag* 2], conteúdo da *tag* 2, continuando sucessivamente. Diante disto, a expressão regular identifica o conteúdo e o *script* de estruturação o associa com a *tag* imediatamente acima, dessa maneira, retornando um *json* com os dados a estruturação das *features* de composição da descrição do relato.

5.2 Coleta dos dados

A API Bugzilla [16] foi usada para coletar relatórios de *bugs* do tipo "*defect*" entre os anos 2013 e 2021, totalizando cerca de 28.284 relatórios. A justificativa para isso é formar uma base atualizada e com a maior dimensão possível, pois existe uma diferença na proporção dos relatos abertos e fechados, o que impossibilitaria a resposta das questões de pesquisa do trabalho. Apenas para exemplificar, dos 9.946 relatos entre 2011 e 2012, apenas 545 foram fechados, cerca de 5,5% do total.

Entretanto, como a postagem dos relatos se trata de um processo manual, é possível que os usuários não preencham os campos texto com nenhum valor, logo, é frequente o valor vazio ("") em *"resolution"* para relatos abertos, conforme o observado no relato 1629538 [20]. Aliado a isso, o nosso *script* produzido para estruturação não é perfeito, ocasionando vez ou outra um valor vazio por não conseguir lidar com marcações especiais de texto, como tabulação sem texto. Contudo, durante o processo de importação do R[5], o valor vazio é tratado como nulo, portanto, nós removemos os dados nulos para permitir o uso de modelos de ML, pois é um requisito do algoritmo.

5.3 Métricas

Na avaliação dos modelos, foram utilizadas as seguintes métricas: *accuracy* [21], *precision* [22], *recall* [22] e *f1-score* (ou *f measure*) [23]. As métricas são calculadas utilizando os valores da matriz de confusão, que destaca os erros e acertos do modelo ao relacionar dados reais e detectados, conforme ilustrado na Tabela 2.

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	True positive	False positive
	Predicted condition negative	False negative	True negative

Tabela 2: Matriz de confusão. Imagem por Júnior [24].

No processo de classificação, a *accuracy* do modelo aponta a performance geral do modelo ao mensurar sua taxa de acerto, também conhecido como resultado verdadeiro, em proporção a todos elementos da base de dados. Porém, devido à construção da sua fórmula, essa métrica não é recomendada quando há um grande desbalanceamento na proporção de resultados positivos e negativos, pois seu resultado dominante pode encobrir o restante. Conforme observado na fórmula abaixo.

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

Por outro lado, as medidas *precision* e *recall* indicam a proporção de resultados positivos (VP) com quantidade de resultados falsos

[5] Plataforma onde foram realizados os experimentos.

positivos (FP) e falsos negativos (FN), respectivamente. Portanto, são métricas complementares para saber a influência de falsos resultados no modelo e, por sua vez, podem ser ajustadas pela média harmônica, também chamada de *F1-score*, que permite maximizar esses valores. Todas com sua representação abaixo.

$$Precision = \frac{VP}{VP + FP} \qquad Recall = \frac{VP}{VP + FN}$$

$$F1\ score = 2 * \frac{(precision * recall)}{precision + recall}$$

5.4 Uso e Configuração do algoritmos

Devido ao número e complexidade de *features* selecionadas, foi necessária uma filtragem para selecionar as *features* candidatas, uma vez que não há recursos computacionais para processar todas elas. A biblioteca MARS [17] foi utilizada para classificar as características mais relevantes, pontuando as relações lineares e logarítmicas em uma métrica de avaliação para os modelos com a variável categórica *"time_class"*. E para isso, esta biblioteca foi aplicada nos *datasets* de relatos abertos (4.706 amostras) e fechados (22.831 amostras), assim, adquirindo uma fórmula específica para construção de conjunto de dados. Logo após, os modelos foram treinados utilizando uma divisão dos dados aleatórios em treino (75%) e teste (25%), que é uma aproximação da proporção recomendada [25] para evitar problemas de *Overfitting*[6]. Na configuração dos modelos, utilizamos a configuração padrão para árvore de decisão e KNN, enquanto que para a *RandomForest* utilizamos o valor 2 na variável *mtry*[7], pois é a recomendação de parâmetros em problemas de classificação no modelo de *RandomForest* [26].

5.5 Ameaças

A principal ameaça interna para esta pesquisa é a estruturação usando expressões regulares, pois se trata de uma solução trivial para lidar com a complexidade da escrita de texto livre e, portanto, não é aplicável na maioria dos casos. Aliada a isso, outra ameaça interna é o nosso *script* produzido para a estruturação do projeto, pois este pode inserir *bugs* que interfiram na elaboração da base de dados pois é responsável por todo controle desse processo. E, por esses motivos, o resultado gerado da estruturação pode não ser suficiente para que sua análise possa gerar alguma informação relevante, visto a quantidade de dados nulos possíveis no *dataset.*

A ameaça externa é sobre as limitações da API Bugzilla [16] que impõe dificuldades no seu uso, como o retorno de apenas 10 mil para relatos para cada consulta e a existência de relatos privados que seu acesso é restringido para consulta externa da API. Portanto, é muito provável que a base de dados não seja o mais completa possível.

6. RESULTADOS E ANÁLISES

6.1 Descrição das Features Candidatas

Ao realizar o experimento, observamos algumas diferenças na seleção de *features* ao aplicar a biblioteca MARS para construção dos modelos de ML. Assim, para relatórios fechados, as *features* candidatas obtidas são: *bugId*, *initialTime*, *endTime*, *comment_count*, *severity*, *priority* e *status*. Logo, em síntese, descrevem informações sobre os identificadores, tamanho da discussão, referências de tempo e status da correção dos relatos. Enquanto os relatos abertos possuem: *comment_count*, *initialTime*, *lastEditTime* e *classification*. Logo, se distinguem pela adição de informações sobre a classificação de origem(cliente, servidor, infraestrutura) e a falta de status da correção nos relatos abertos, o que condiz com a definição da classificação entre relatos abertos e fechados. Além disso, podemos indicar que a complexidade dos modelos de relatórios abertos é menor que dos fechados dado a quantidade de features que os compõem.

6.2 Performance dos modelos

Em relação aos relatos fechados, o modelo *RandomForest* se destacou por apresentar uma *accuracy* de 64% e não exibir nenhuma discrepância na influência de resultados falsos, pois suas outras métricas apresentaram um resultado acima de 53%, o melhor observado entre todos os modelos. Conforme ilustrado na Tabela 3.

	model	.metric	.estimator	.estimate
1	DecisionTree	precision	macro	0.4526347
2	DecisionTree	recall	macro	0.2925792
3	DecisionTree	accuracy	multiclass	0.4358795
4	DecisionTree	f_meas	macro	0.5498380
5	KNN	precision	macro	0.4932999
6	KNN	recall	macro	0.4813765
7	KNN	accuracy	multiclass	0.5632446
8	KNN	f_meas	macro	0.4861117
9	RandomFlorest	precision	macro	0.6400171
10	RandomFlorest	recall	macro	0.5318620
11	RandomFlorest	accuracy	multiclass	0.6532936
12	RandomFlorest	f_meas	macro	0.5473970

Tabela 3 - Métricas dos modelos de relatos fechados

De acordo com a Tabela 3, é visível que o modelo de *DecisionTree* obteve resultados inferiores aos outros modelos e a causa disso é que sua execução no *dataset* de teste não conseguiu classificar variáveis das classes: D11_30, D31_60 e DX_1, segundo *warning* no ambiente R, embora as classes estivessem presentes do subconjunto de teste. Por este motivo, concluímos que o algoritmo não foi adequado para estes conjuntos de dados e que os mecanismos internos do modelo realizaram a remoção dessas classes na matriz de confusão, prejudicando as métricas do modelo. Na Tabela 4, são exibidas todas as linhas da predição onde o modelo não conseguiu realizar nenhuma classificação na sua matriz de confusão, demonstrando a remoção das classes citadas.

[6] Sobre ajuste. Descreve um modelo que se ajusta muito bem aos dados de treinamento mas é ineficaz para trabalhar com novos dados

[7] Número de variáveis amostradas aleatoriamente como candidatas em cada divisão.

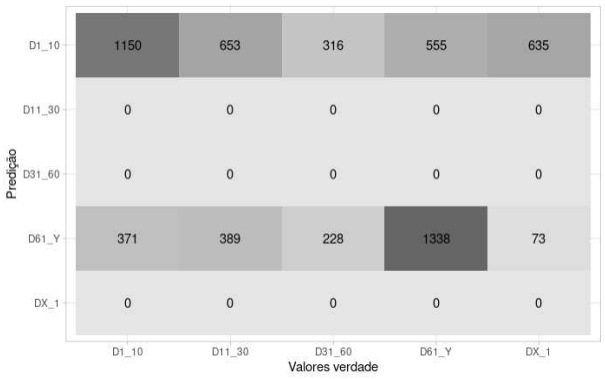

Tabela 4 - Matriz de confusão da DecisionTree dos relatos fechados.

Em relação aos relatos abertos, apesar dos modelos apresentarem uma *accuracy* de 86-92%, a maioria dos modelos não são confiáveis, pois apresentam *recall* ou *precision* abaixo das suas respectivas acurácias, o que indica forte presença de falsos positivos e falsos negativos na matriz de confusão, com a única exceção sendo o modelo de *RandomForest*. Conforme ilustrado na Tabela 5.

	model	.metric	.estimator	.estimate
1	DecisionTree	accuracy	multiclass	0.8689076
2	DecisionTree	f_meas	macro	0.8135260
3	DecisionTree	precision	macro	0.7316195
4	DecisionTree	recall	macro	0.3958740
5	KNN	accuracy	multiclass	0.8630252
6	KNN	f_meas	macro	0.4806379
7	KNN	precision	macro	0.4794708
8	KNN	recall	macro	0.4845274
9	RandomFlorest	accuracy	multiclass	0.9226891
10	RandomFlorest	f_meas	macro	0.6959357
11	RandomFlorest	precision	macro	0.7278728
12	RandomFlorest	recall	macro	0.6795532

Tabela 5 - Métricas dos modelos de relatos abertos

Por fim, concluímos que os modelos treinados de ML alcançaram um resultado melhor na predição da classificação dos relatos abertos, apesar das ressalvas destacadas acima. Neste resultado, podemos responsabilizar a diferença da complexidade e a quantidade de amostras de testes como os fatores determinantes para de desempenho entre os modelos de abertos e fechados, com destaque para o modelo KNN, que necessita de um aumento exponencial de amostras para manter uma boa performance à medida que a complexidade aumenta [31].

6.3 Importância das features

Os algoritmos de aprendizagem supervisionada já possuem mecanismos internos que podem mensurar a importância das *features* utilizadas, por isso foi selecionado para esta análise os modelos de *RandomFlorest* pois apresentaram os melhores resultados nas métricas da pesquisa. Logo, utilizando a interface da biblioteca "vip" [27], o resultado obtido foi esperado com as *features* relacionadas ao tempo (*endTime*, *lastEditTime*, *initialTime*), visto que são as variáveis em que tempo de resolução é derivado e, como tal, é um entendimento pressuposto sobre a análise. Assim, esperava-se a maior importância dessas *features*, conforme ilustrado pela Figura 2.

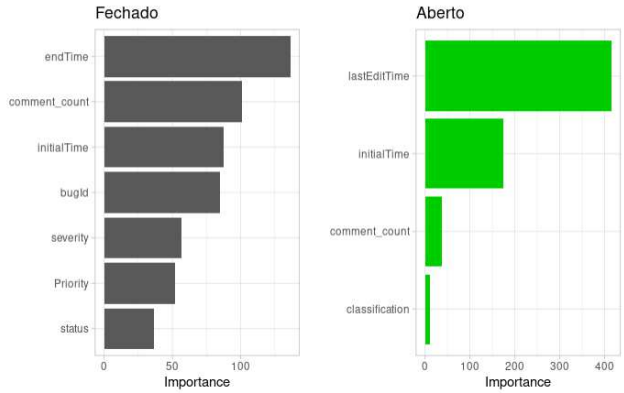

Figura 2 - Resultado da análise da importância das variáveis.

Como descreve a Figura 2, a feature *comment_count* está presente em ambos os tipos de relatos, o que a torna um excelente comparador para análises futuras. Além disso, é inesperado que o identificador (*bugId*) seja tão significativo quanto a feature de tempo (*initialTime*) para os modelos de relatos fechados. A seção aprofunda um pouco mais a discussão sobre esta feature.. Por fim, a influência de *classification* é quase inexistente, logo, foi descartado para análise descritiva deste estudo.

6.4 Influências dos comentários

Em relação aos relatos fechados, as soluções com duração de até 30 dias (as classes DX_1; D1_10; D11_30) são mais frequentes no intervalo de até 25 comentários. E, com aumento crescente dos comentários, as classes mais de 60 dias (D61_X) e entre 31 a 50 dias (D31_60) vão se tornando mais presentes. Portanto, interpretamos que a quantidade de comentários indica um aumento do tempo de resolução do relatório quando passamos do limite de 25 comentários, independente do número de pessoas envolvidas na interação, dado as *features* avaliadas nesta pesquisa. A Figura 3 demonstra que as classes (DX_1; D1_10; D11_30) representam um valor equivalente ou maior que as classes (D61_X e D31_60) ao somar suas frequências relativas até o eixo X representa 25 comentários, após, a inversão dessa proporção.

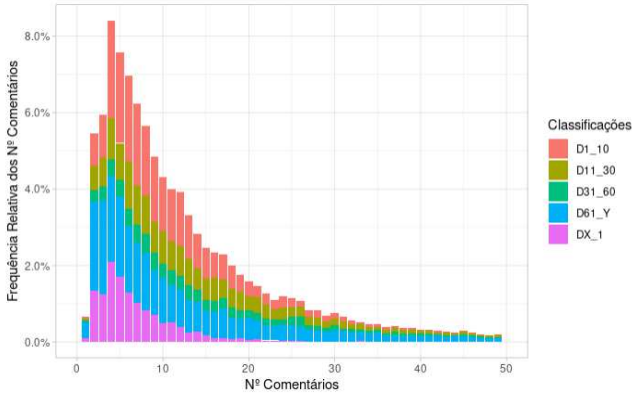

Figura 3: Frequência relativa no Nº de Comentário do *dataset* dos fechados.

Por outro lado, os relatos abertos apontam que um pequeno número de comentários está relacionado com abandono, ou falta de conclusão, em discussões de mais de 60 dias (D61_Y) ou no mesmo dia (DX_1). E, para o segundo caso, nunca há qualquer feedback de outro usuário da plataforma [5], como no relato 934920 [30]. Assim, analisando a frequência relativa de comentários, percebemos que o maior número dessas classes ocorre quando a postagem possui menos de 10 comentários. Dessa maneira, podemos concluir que um intervalo ideal de comentários é em torno de 10-25 comentários.

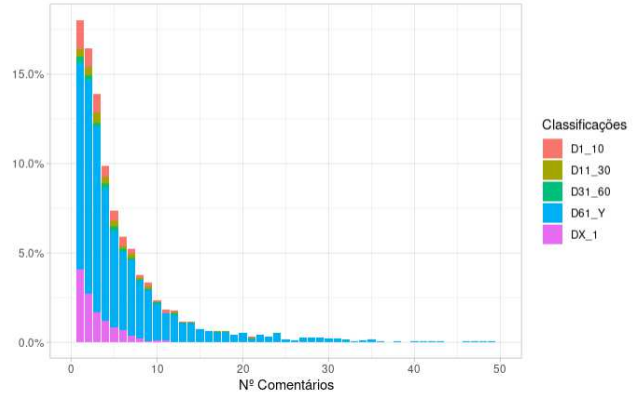

Figura 4: Frequência relativa no Nº de Comentário do *dataset* abertos.

Para contextualizar, podemos notar que no relato 1558070 [29], a equipe de suporte teve dificuldade de reproduzir o relato pela mensagem "*Thanks! No luck reproducing the issue yet, unfortunately…*" aparecer quase no final da discussão. Anterior a isto, a preocupação do time de desenvolvimento era adquirir o máximo de informações, como configurações de variáveis, sistemas operacionais afetados, passos para reprodução, etc. Com a publicação de "*Changing the priority to p1 as the bug is tracked by a release manager for the current release.*" alterar a mudança de prioridade(P3->P1) e indicar que o *bug* foi monitorado, sendo necessário apenas mais 14 comentários e 11 dias para solucionar o relato, o intervalo ideal proposto acima. Desse modo, concluímos ser possível diminuir os 67 dias e 43 comentários gastos caso a postagem do relato possuísse todas as informações necessárias.

E para corroborar com as conclusões acima, há uma tendência entre o aumento de prioridade e a diminuição da duração do tempo, conforme o histograma abaixo e o ocorrido no relato 1558070 [29], com esta distribuição muito semelhante entre P1 e P2 nos relatos fechados. Além disso, como quase não existem relatos abertos a outras categorias de prioridade [19], podemos identificar a prática da plataforma de manter sem prioridade ("--") os relatos abertos ou que não estão em processo de solução, com base no grupo à direita do histograma, ilustrado na Figura X.

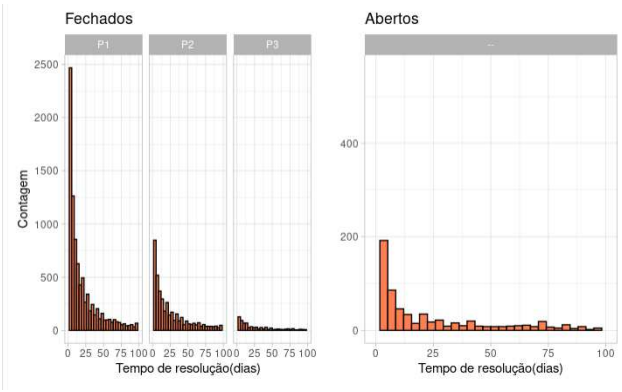

Figura 5: Histograma com a frequência do tempo de resolução dos relatos, divididos em prioridades.

6.5 Estruturação sem resultados

Conforme discutido nas ameaças, o uso experimental de expressão regular obteve muitos resultados nulos na estruturação da base de dados, o que gerou 94-100% de dados inexistentes nessas *features*. Assim, nenhuma das *features* resultantes apresentou resultados na seleção de *features* [18] e muito menos no treinamento dos modelos que, por sua vez, impossibilita qualquer analíse mais profunda sobre a estrutura da escrita de relatórios de *bugs*, que poderia ser realizado com técnicas de tokenização, relevância de termos, etc. Portanto, concluímos que o processo de estruturação foi uma limitação de metodologia nesta pesquisa.

6.6 Tendência da plataforma

A princípio, o *bugId* é *feature* para identificar unicamente um relato, porém, ela também é um vestígio da evolução da plataforma por ser uma variável crescente ao passar do tempo, sendo um fato que relatos mais recentes possuem um maior ID. Portanto, analisando a relação linear entre *bugId* e tempo de resolução em dias, observamos que há uma tendência na redução do tempo de resolução com o passar do tempo, representado pelo feature *bugId*.

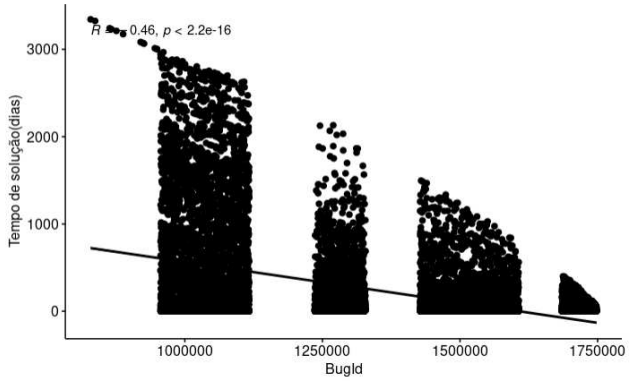

Figura 6: Relação linear entre *bugId* e tempo de solução.

Apesar da Figura 6 apresentar uma correlação linear de 0.46(R) entre tempo de resolução e *bugId*, o resultado não é estatisticamente significativo por causa que o valor p é bem menor a um nível de significância de 5%, padrão utilizado para essas avaliações. Contudo, mesmo não representando uma correção linear significativa, a figura indica uma diminuição do tempo de resolução em cada agrupamento de valores que a reta passa, o que não pode ser desconsiderado apenas pela correlação linear.

Por fim, uma explicação para esta tendência é uma maior atividade no suporte da plataforma nos últimos anos. Pois, comparando datasets da plataforma [5], temos que em 2011 à 2012 os relatos fechados representam 5,5% do total, enquanto nos anos de 2013 e 2021 essa proporção aumentou para 81%. Portanto, indicando que a equipe de suporte teve maior participação nos anos mais recentes da plataforma, porque é a equipe de suporte responsável por fecharem (concluírem) os relatos.

6.7 Outras features candidatas

Embora a importância das *features* indicarem como significativo as *features severity* e *status*, nenhuma informação relevante pôde ser extraída da uma análise estatística descritiva. Dessa forma, é explicação para isso as diferenças entre valores assumidos entre os relatos fechados e abertos, por isso, tornando possível processo de classificação baseado na presença dessas informações. Conforme a Figura 6.

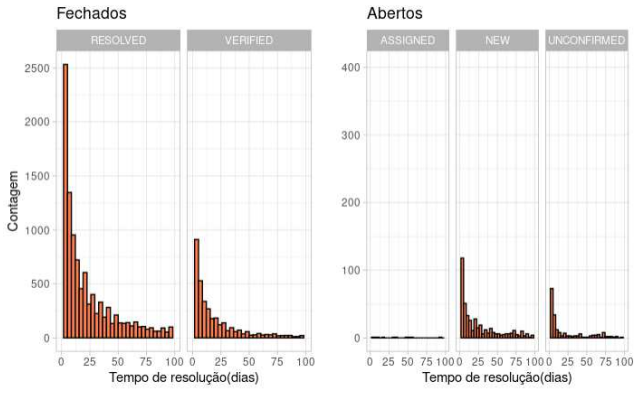

Figura 7: Histograma do tempo de resolução separado por categorias de status.

6.8 Retomando as questões de pesquisa

Estabelecido os resultados e as implicações da sua análise, podemos responder às questões da pesquisa.

6.8.1 Quais são os principais parâmetros (features) que influenciam no tempo de resolução do relatório de bugs fechados? (QP1)

Com base na nossa análise, temos que as *features* mais representativas são: prioridade, quantidade de comentários e quão recente é o relato. Dessa forma, quando o relato é definido com uma alta prioridade, como P1 e P2, há o aumento da probabilidade de conclusão do relato. A causa disso é a política de triagem, plataforma que prioriza os esforços da equipe de suporte para atividades com mais prioridades [32]. Por outro lado, quando um relato possui poucos comentários (menos de 25), a uma maior tendência deles serem fechados em menos de 30 dias, provavelmente indicando que relatos bem escritos não necessariamente precisam de muitos comentários. Por fim, na plataforma os relatos mais recentes possuem menor número de comentários e, por implicação, também aumentam as chances dele ser concluído.

6.8.2 Qual é a principal diferença entre relatórios de bugs abertos e fechados? (QP2)

As *features* responsáveis por diferenciar os relatos abertos e fechados são: quantidade de comentários e informações de correção. Em contraposto aos relatos fechados, quando relatos possuem 7 ou menos comentários a uma predisposição para que sejam mantidos em execução(abertos) ao ponto que nunca podem ter sido revisados por outros usuários da plataforma. Logo, criando um intervalo ideal entre 8 a 25 comentários para a solução do relatório quando este não possui nenhum problema, como informações imprecisas, faltantes ou ambíguas. Além disso, as informações de correção *(status, priority)* se diferem entre relatos abertos e fechados, o que faz total sentido, já que esses relatos estão em níveis diferentes de correção do bug.

7. CONCLUSÃO

Este trabalho é um estudo quantitativo que buscou identificar fatores que influenciam o processo de resolução dos relatórios de bug provenientes da API do Bugzilla. Neste processo, aconteceu uma limitação ao estruturar relatos em subcampos mais descritivos por meio de expressão regular, o que ocasionou apenas novas *features* nulas. Como resultado, confirmamos uma tendência no aumento da atividade da equipe de suporte do Bugzilla e encontramos um intervalo ideal de comentários para resolução de *bugs*, entre 8 a 25, que pode ser utilizado por desenvolvedores como métrica de qualidade no processo de resolução de *bugs*. Por fim, para trabalhos futuros, pode-se refazer este estudo utilizando abordagem qualitativa ou ferramentas sofisticadas para estruturação dos relatórios.

8. AGRADECIMENTOS

Ao meu orientador, prof. Franklin Ramalho, pela orientação e dedicação nesse estudo. Aos colaboradores, Thaís Toscano, José Ferreira e o prof. prof. Massoni, pelas sugestões e o apoio na pesquisa. E aos meus amigos e familiares por sempre me apoiarem na minha jornada.

9. REFERÊNCIAS

[1] Sommerville, I. 2011. Engenharia de Software. Pearson, 9ª. Edição. Pg 475-478. Acesso em 27/09/21.

[2] Battenburg, N. 2010. Extracting structural information from bug reports. Pg. 1-3. Disponível no <https://www.researchgate.net/publication/221656988_Extracting_structural_information_from_bug_reports>. Acesso em 30/09/21.

[3] Davies, S. 2014. What 's in a Bug Report?. 1-2 pg. Disponível em <https://dl.acm.org/doi/10.1145/2652524.2652541>. Acesso em 08/08/21.

[4] Battenburg, N. 2010. What Makes a Good Bug Report?. Pg. 1-6. Disponível em <https://dl.acm.org/doi/10.1145/2652524.2652541>. Acesso em 10/08/21.

[5] Bugzilla, plataforma web de bug report da Mozilla. Disponível no <https://bugzilla.mozilla.org/home>. Acessado 09/10/21

[6] Raymond, E.S. 2001. The Cathedral and The Bazaar. Edição revisada. Pg. 30-33.Disponível em <https://monoskop.org/images/e/e0/Raymond_Eric_S_The_Cathedral_and_the_Bazaar_rev_ed.pdf>. Acesso em 27/09/21.

[7] Yui, T. 2019. Artigo “Understanding Random Forest”, Disponível em <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

[8] Decision Tree. 2012. Artigo do GeekForGeeks. Disponível no <https://www.geeksforgeeks.org/decision-tree/>. Acessado em 27/03/22

[9] Harrison, O. 2018. Machine Learning Basics with the K-Nearest Neighbors Algorithm. Disponível em <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>. Acessado em 26/03/22

[10] Shwartz, S. S. 2014. “Understanding Machine Learning: From Theory to Algorithms”. Versão Digital. Pg 7-8. Disponível em <https://www.cs.huji.ac.il/w~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf>

[11] Mitchell, T. M. 1997. “Machine Learning”, 1 edição. Editora McGraw-Hill Education. Pg 14.

[12] Haykin S, 2008; “Neural Networks and Learning Machines” 3rd Edition. Prentice Hall. Pg 36-37. Disponível em <https://lps.ufrj.br/~caloba/Livros/Haykin2009.pdf>

[13] Henke, M. 2011. “Aprendizagem de Máquina para Segurança em Redes de Computadores: Métodos e Aplicações”, Livro de Minicursos do XI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. 1a Edição. Pg 12-23. Disponível em <https://www.researchgate.net/publication/228447003_Aprendizagem_de_Maquina_para_Seguranca_em_Redes_de_Computadores_Metodos_e_Aplicacoes>. Acessado 15/03/22.

[14] Anvik, Josn. 2006. Who Should Fix This Bug? Pg 1-2 . Disponível em <https://doi.org/10.1145/1134285.1134336> Acessado em 04/03/22.

[15] Rodrigues, E.M. 2021. Técnicas de Machine Learning para Predição do Tempo de Permanência na Graduação no mbito do Ensino Superior Público Brasileiro. Acesso em 10/02/22.

[16] API do Bugzilla. Documentação oficial. Disponível em: <https://bmo.readthedocs.io/en/latest/using/index.html>. Acesso em 15/02/22.

[17] MARS. Artigo Multivariate Adaptive Regression Splines. Disponível em <http://uc-r.github.io/mars>. Acesso em 25/03/22.

[18] Feature Selection. Wikipédia. Disponível no <https://en.wikipedia.org/wiki/Feature_selection>. Acesso em 25/03/22

[19] Priority System Bugzilla. Wiki da ferramenta. Disponível no <https://wiki.mozilla.org/Bugzilla:Priority_System>. Acesso em 25/03/22

[20] Bugzilla. Postagem 1629538 da plataforma. Disponível em <https://bugzilla.mozilla.org/show_bug.cgi?id=1629538>. Acesso em 25/03/22.

[21] Accuracy, Course Machine Learning - Google Developers. Disponível em <https://developers.google.com/machine-learning/crash-course/classification/accuracy>. Acesso em 15/03/22.

[22] Recall and Precision, Course Machine Learning - Google Develops. Disponível em <https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall>. Acesso em 15/03/22.

[23] F1-Score. Wikipédia. Disponível em: <https://en.wikipedia.org/wiki/F-score>. Acesso em 15/03/22.

[24] Júnior, D.C. 2018. Imagem retirada de “Classificação de editais licitatórios em áreas de atuação baseado em aprendizado supervisionado”. Pg 31. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/192305/TCC%20%287%29.pdf?sequence=1&isAllowed=y>. Acesso em 30/03/22.

[25] Branco, H. 2020. Artigo “Overfitting e underfitting em Machine Learning”. Disponível em <https://abracd.org/overfitting-e-underfitting-em-machine-learning>. Acesso em 27/03/22.

[26] Boehmke, B. Livro “Hands-On Machine Learning with R”, Capítulo 11. Disponível em <https://bradleyboehmke.github.io/HOML/random-forest.html>. Acesso em 24/03/22.

[27] Bugzilla. Postagem 1558070 da plataforma Bugzilla. Disponível em <https://bugzilla.mozilla.org/show_bug.cgi?id=1558070>. Acesso em 22/03/22.

[28] Biblioteca “vip”, Variable importance plots. Disponível em: <https://koalaverse.github.io/vip/articles/vip.html>. Acesso em 27/03/22.

[29] Bugzilla. Postagem 1558070 da plataforma Bugzilla. Disponível em

<https://bugzilla.mozilla.org/show_bug.cgi?id=1558070>. Acesso em 25/03/22.

[30] Bugzilla. Postagem 934920 da plataforma Bugzilla. Disponível em <https://bugzilla.mozilla.org/show_bug.cgi?id=934920>. Acesso em 27/03/22.

[31] O Algorítmo k-Nearest Neighbors (kNN) em Machine Learning. 2018. Artigo do PortalDataScience. Disponível em <https://portaldatascience.com/o-algoritmo-k-nearest-neighbors-knn-em-machine-learning/>. Acesso 23/03/22.

[32] Politicas de Tiragem da plataforma Bugzilla, Disponível em <https://firefox-source-docs.mozilla.org/bug-mgmt/policies/triage-bugzilla.html> Acesso em 22/03/22.