UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MATHIAS ABREU TRAJANO

UMA ABORDAGEM PARA CLASSIFICAÇÃO DE COMENTÁRIOS E ANÁLISE DE IMPACTO NO TEMPO DE RESOLUÇÃO EM RELATÓRIOS DE BUGS: UM ESTUDO DE CASO NO BUGZILLA

CAMPINA GRANDE – PB
2026

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem para Classificação de Comentários e Análise de Impacto no Tempo de Resolução em Relatórios de Bugs: Um Estudo de Caso no Bugzilla

Mathias Abreu Trajano

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande – Campus 1 como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Franklin de Souza Ramalho e Tiago Lima Massoni
(orientadores)

Campina Grande, Paraíba, Brasil

Universidade Federal de Campina Grande - UFCG
Sistema de Bibliotecas - SISTEMOTECA
Catalogação de Publicação na Fonte. UFCG - Biblioteca Central

A663r Trajano, Mathias Abreu.

Uma abordagem para classificação de comentários e análise de impacto no tempo de resolução em relatórios de bugs: um estudo de caso no Bugzilla / Mathias Abreu Trajano. – 2026.

165 f. ; il. color.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2026.

"Orientação: Prof. Dr. Tiago Lima Massoni, Prof. Dr. Franklin de Souza Ramalho".

Referências.

1 Engenharia de Software. 2. Bugzilla. 3. Machine Learning. 4. Taxonomia. 5. Relatórios de Bugs. I. Massoni, Tiago Lima. II. Ramalho, Franklin de Souza. III. Título.

UFCG/BC CDU 004.41(043.3)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
POS-GRADUACAO EM CIENCIA DA COMPUTACAO
Rua Aprígio Veloso, 882, Edifício Telmo Silva de Araújo, Bloco CG1, - Bairro Universitário, Campina Grande/PB, CEP 58429-900
Telefone: 2101-1122 - (83) 2101-1123 - (83) 2101-1124
Site: http://computacao.ufcg.edu.br - E-mail: secpg@computacao.ufcg.edu.br

FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

MATHIAS ABREU TRAJANO

UMA ABORDAGEM PARA CLASSIFICAÇÃO DE COMENTÁRIOS E ANÁLISE DE IMPACTO NO TEMPO DE RESOLUÇÃO EM RELATÓRIOS DE BUGS: UM ESTUDO DE CASO NO BUGZILLA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 05/02/2026

Prof. Dr. TIAGO LIMA MASSONI, Orientador, UFCG

Prof. Dr. FRANKLIN DE SOUZA RAMALHO, Orientador, UFCG

Prof. Dr. EVERTON LEANDRO GALDINO ALVES, Examinador Interno, UFCG

Prof. Dr. LINCOLN SOUZA ROCHA, Examinador Externo, UFC

Documento assinado eletronicamente por **EVERTON LEANDRO GALDINO ALVES**, **PROFESSOR 3 GRAU**, em 27/03/2026, às 15:19, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **TIAGO LIMA MASSONI**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 29/03/2026, às 11:08, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **FRANKLIN DE SOUZA RAMALHO**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 30/03/2026, às 09:53, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **LINCOLN SOUZA ROCHA**, **Usuário Externo**, em 14/04/2026, às 08:37, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

A autenticidade deste documento pode ser conferida no site https://sei.ufcg.edu.br/autenticidade, informando o código verificador **6371161** e o código CRC **6A1D371E**.

Referência: Processo nº 23096.006665/2026-58 SEI nº 6371161

Resumo

Relatórios de bugs são importantes fontes de informações sobre problemas que ocorrem no desenvolvimento de software. Diversas linhas de pesquisa são voltadas para a compreensão desses relatórios e como eles podem ajudar a comunidade de desenvolvimento de software. Neste estudo, propomos uma investigação em relatórios de bugs com uma ênfase voltada para a análise de comentários desses relatórios. Procuramos obter informações acerca da influência dos comentários no tempo de resolução dos mesmos. Para isso definimos uma taxonomia de classes de comentários construída a partir da análise qualitativa do conteúdo textual e dos tipos de anexos associados, com o objetivo de representar as formas mais recorrentes de contribuição observadas durante o ciclo de vida dos relatórios de bugs. Com base na taxonomia, foram treinados e avaliados modelos de Machine Learning para a classificação automática dos comentários. Os experimentos demonstram que os modelos são capazes de classificar os comentários de forma consistente entre as classes propostas, permitindo a identificação de padrões entre os tipos de comentários e os tempos de resolução dos relatórios. Os resultados indicam que determinadas classes de comentários estão associadas a resoluções mais rápidas, enquanto outras classes se relacionam a maiores tempos de resolução, evidenciando a relevância da classificação de comentários como ferramenta de apoio à análise do processo de manutenção de software.

Abstract

Bug reports are important sources of information about problems that occur during software development. Several lines of research are focused on understanding these reports and how they can help the software development community. In this study, we propose an investigation of bug reports with an emphasis on the analysis of their comments. We seek to obtain information about the influence of comments on their resolution time. To this end, we define a taxonomy of comment classes built from a qualitative analysis of textual content and associated attachment types, with the objective of representing the most recurrent forms of contribution observed during the bug report lifecycle. Based on the taxonomy, Machine Learning models were trained and evaluated for the automatic classification of comments. The experiments demonstrate that the models are able to classify comments consistently among the proposed classes, allowing the identification of patterns between comment types and bug resolution times. The results indicate that certain classes of comments are associated with faster resolutions, while other classes are related to longer resolution times, highlighting the relevance of comment classification as a supporting tool for the analysis of the software maintenance process.

Dedicatória

Dedico esta dissertação, em primeiro lugar, à memória de meu primo Jânio Benedito de Souza, cuja partida precoce deixou um vazio impossível de ser medido, mas também uma lembrança que o tempo não apaga. Sua ausência se transformou em saudade, e a saudade, em presença silenciosa dentro de tudo aquilo que carrego comigo.

Dedico este trabalho aos meus pais, Edcarla Abreu e José Adailton, que são o princípio de tudo o que sou. Foram eles que, com amor, sacrifício e coragem, abriram caminhos onde antes havia incerteza, e sustentaram meus passos mesmo quando o peso da vida lhes exigia tanto. Se hoje chego até aqui, é porque antes de mim houve mãos que construíram, ombros que suportaram, corações que acreditaram. Foram vocês que enfrentaram o sol para que eu pudesse encontrar sombra, descanso e possibilidade.

À minha avó materna, Maria da Conceição, por toda a ternura, cuidado e sabedoria que sempre me envolveram; e à memória de meu avô materno, Francisco Pereira, cuja lembrança permanece viva em minha história como herança de afeto, dignidade e ensinamento.

Aos meus avós paternos, José Trajano Borges e Maria do Socorro, por representarem as raízes que me sustentam e a história da qual me orgulho de fazer parte.

Dedico ainda este trabalho a todos os meus tios, por cada gesto de apoio ao longo da caminhada, e às famílias Abreu e Trajano, que são o chão afetivo da minha existência, o lugar de onde venho e a força que me acompanha.

Agradecimentos

A Deus, pela vida e pelo fôlego de cada dia. Pela força que me sustentou nos momentos em que fraquejei, pela esperança que renasceu quando o caminho parecia difícil demais, e pela graça que me permitiu chegar até aqui. Em muitos momentos, foi a fé que me manteve de pé.

Aos meus pais, Edcarla Abreu e José Adailton, dirijo minha mais profunda gratidão. Não há palavra que alcance plenamente tudo o que vocês representam em minha vida. Esta conquista não começou em mim: ela começou no amor de vocês, nos esforços muitas vezes invisíveis, nos silêncios cheios de renúncia, nas preocupações guardadas, nas orações, no trabalho duro e na esperança depositada em meu futuro. Vocês sonharam comigo quando eu ainda nem sabia sonhar direito. Vocês acreditaram em mim quando o caminho ainda era longo. Tudo o que sou leva a marca do amor de vocês. Esta dissertação, mais do que minha, também é de vocês.

Ao meu primo Jânio Benedito de Souza, in memoriam, pela lembrança que permanece viva e pela saudade que, embora dolorosa, também me ensina sobre o valor das pessoas, da família e dos laços que o tempo não desfaz. Há ausências que nunca deixam de doer, mas que continuam iluminando a memória com amor.

À minha avó materna, Maria da Conceição, pelo afeto constante, pelo cuidado, pelas palavras e pela presença que tantas vezes significaram abrigo. Ao meu avô materno, Francisco Pereira, in memoriam, cuja memória segue viva em meu coração e em minha trajetória. Aos meus avós paternos, José Trajano Borges e Maria do Socorro, pela história, pelos exemplos de vida e pelo legado que atravessa gerações e chega até mim como força, pertencimento e identidade.

A todos os meus tios, agradeço pelo apoio, pela generosidade e pela presença em diferentes momentos da minha caminhada. Cada ajuda, cada palavra e cada gesto de cuidado tiveram um valor que talvez nem sempre tenha sido dito, mas que sempre foi profundamente sentido.

Às famílias Abreu e Trajano, meu sincero agradecimento por serem parte essencial da minha história. Em cada vínculo familiar encontrei força, memória, amparo e o sentimento de que nunca caminhei sozinho. Sou fruto de uma história coletiva, tecida por muitas mãos, muitos afetos e muitas lutas.

À comunidade de Cajazeiras Velha, deixo minha gratidão e meu carinho. A esse lugar devo memórias, vivências, ensinamentos e marcas afetivas que ajudaram a moldar quem sou. Há lugares que não são apenas espaços, mas parte da alma da gente; Cajazeiras Velha é, para mim, um desses lugares que permanecem vivos dentro do coração.

Aos meus orientadores, Tiago Massoni e Franklin Ramalho, agradeço profundamente pela generosidade intelectual, pela confiança, pela escuta e pela orientação cuidadosa ao longo desta trajetória. Obrigado pelas leituras atentas, pelas contribuições valiosas, pelo rigor acadêmico e pela condução respeitosa e comprometida deste trabalho. Mais do que orientar uma pesquisa, vocês contribuíram para minha formação como pesquisador e para meu amadurecimento ao longo deste percurso.

Ao SPLab, pela estrutura, pelo suporte e pelo ambiente que possibilitou o desenvolvimento desta pesquisa. Sou grato pelo acolhimento institucional e pelas condições oferecidas para que este trabalho pudesse se concretizar.

Ao professor Andrey, a Fellype e ao LSD (Laboratório de Sistemas Distribuídos), meu agradecimento sincero pelo apoio, pela receptividade e pela colaboração ao longo dessa caminhada. Toda ajuda recebida foi importante para tornar este percurso possível.

Aos meus colegas de mestrado, em especial Sheila Maria e Diego, agradeço pelo companheirismo, pelo incentivo e pela partilha das alegrias e dificuldades que marcaram esta etapa. Caminhar ao lado de pessoas que compreendem os desafios do percurso faz toda a diferença, e vocês tornaram essa jornada mais leve e mais humana.

Aos meus colegas da LiteMe, pela convivência, pelo apoio e pelas trocas que, de diferentes formas, contribuíram para minha formação profissional e pessoal. Cada experiência compartilhada também deixou marcas importantes nesta trajetória.

Aos meus amigos, que são tantos e tão fundamentais em minha vida, deixo meu mais afetuoso agradecimento. São muitos os nomes, muitos os rostos, muitas as histórias, e em todos eles encontro razões para agradecer. Obrigado pela amizade sincera, pela escuta paciente, pelas palavras de incentivo, pelo riso nos dias difíceis e pela presença que tantas vezes foi abrigo. Em meio ao cansaço, às dúvidas e às

exigências do caminho, foram vocês que me lembraram, de diferentes formas, que nenhuma jornada precisa ser atravessada em solidão.

Por fim, agradeço a todos aqueles que, de forma direta ou indireta, fizeram parte desta caminhada. Toda conquista individual carrega dentro de si a presença de muitos: daqueles que apoiaram, daqueles que ensinaram, daqueles que torceram, daqueles que permaneceram. Esta dissertação é o resultado de estudo, esforço e persistência, mas também é, profundamente, o resultado do amor, da confiança e da presença de tantas pessoas que caminharam comigo até aqui.

Sumário

1 Introdução 1
1.1 Motivação 1
1.2 Problema 4
1.3 Objetivos 6
1.4 Questões de Pesquisa 7
1.5 Escopo 8
1.6 Contribuições 9
1.7 Estrutura do Documento 9
2 Fundamentação Teórica 10
2.1 Relatórios de Bugs 10
2.2 Bugzilla 12
2.3 Machine Learning 15
3 Trabalhos Relacionados 17
4 Metodologia 21
4.1 Visão Geral 21
4.2 Coleta de dados 23
4.2.1 Obter Features dos relatórios de bugs 24
4.2.2 Obter comentários e histórico de modificações 25
4.3 Caracterização dos Dados 28
4.4 Processamento de Comentários 30
4.4.1 Processamento de Texto 30
4.5 Classes de Comentários 32
4.6 Experimento de classificação de comentários 34
4.6.1 Modelos Supervisionados 36
4.6.2 Modelos Não Supervisionados 37
4.6.3 LLMs (Large Language Models) 38
4.7 Variações de Dados 39
4.8 Experimento de Previsão da Duração de Resolução de Relatórios de bugs 40
4.9 Ajustes de Hiperparâmetros 43
4.10 Avaliação e Teste 44
4.11 Ameaças a Validade 48
5 Taxonomia 49
5.1 Definição das Classes de Comentários e Anexos 49
5.1.1 Classes 51

5.2 Considerações sobre a Definição e Aplicação das Classes de Comentários e Anexos 74
6 Classificação Manual dos Comentários 76
6.1 Análises sobre os Comentários classificados manualmente 77
6.2 Análises da Distribuição das Classes de Comentários e Anexos 81
7 Resultados do Experimento de classificação de comentários 84
7.1 Modelos Supervisionados 84
7.1.1 Panorama geral dos resultados dos modelos supervisionados 97
7.1.2 Análise por classes de comentários 104
7.2 Modelos Não Supervisionados 115
7.2.2 Resultados Gerais dos modelos não supervisionados 124
7.3 LLMs 127
7.3.1 Limitações do Experimento com LLMs e Planos Futuros 130
8 Resultados do Experimento de Previsão da Duração de Resolução de Relatórios de bugs 131
8.1 Modelos de Regressão 132
8.2 Modelos de Classificação Supervisionada 136
9 Conclusões 150
9.1 Respostas às Questões de Pesquisa 150
9.2 Limitações do Trabalho 153
9.3 Implicações 154
9.3.1 Implicações para a pesquisa em Relatórios de Bugs 154
9.3.2 Implicações para engenheiros de software 155
9.4 Contribuições do Trabalho 156
9.5 Trabalhos Futuros 157
Referências 159

1 Introdução

Este capítulo descreve inicialmente sobre a motivação e o contexto do estudo, discorrendo sobre informações cruciais como o objetivo proposto, o problema a ser a analisado, questões de pesquisa consideradas, limitações de escopo e, por fim, as contribuições do estudo.

1.1 Motivação

Relatórios de bugs são fontes poderosas para obter e disseminar informações acerca de problemas que estão sendo encontrados em softwares [1] e sistemas de informações [2]. Eles desempenham um papel fundamental no desenvolvimento de software, proporcionando um canal vital para identificar e corrigir falhas em um sistema. Esses relatórios detalhados capturam anomalias e comportamentos indesejados em softwares, fornecendo informações cruciais para desenvolvedores e testadores. Um relatório de bug possui no seu corpo campos informacionais para determinar todos os detalhes do problema relatado, sendo os principais: o título, que fornece um resumo limitado sobre o problema; a descrição, que explica detalhadamente o bug incluindo informações de como o problema ocorreu, circunstâncias envolvidas e qual era o comportamento esperado; os passos para reprodução, contendo o passo a passo das ações necessárias para reproduzir o bug; e o campo de anexos que permite a adição de capturas de tela e outros arquivos que podem ajudar na descrição do bug.

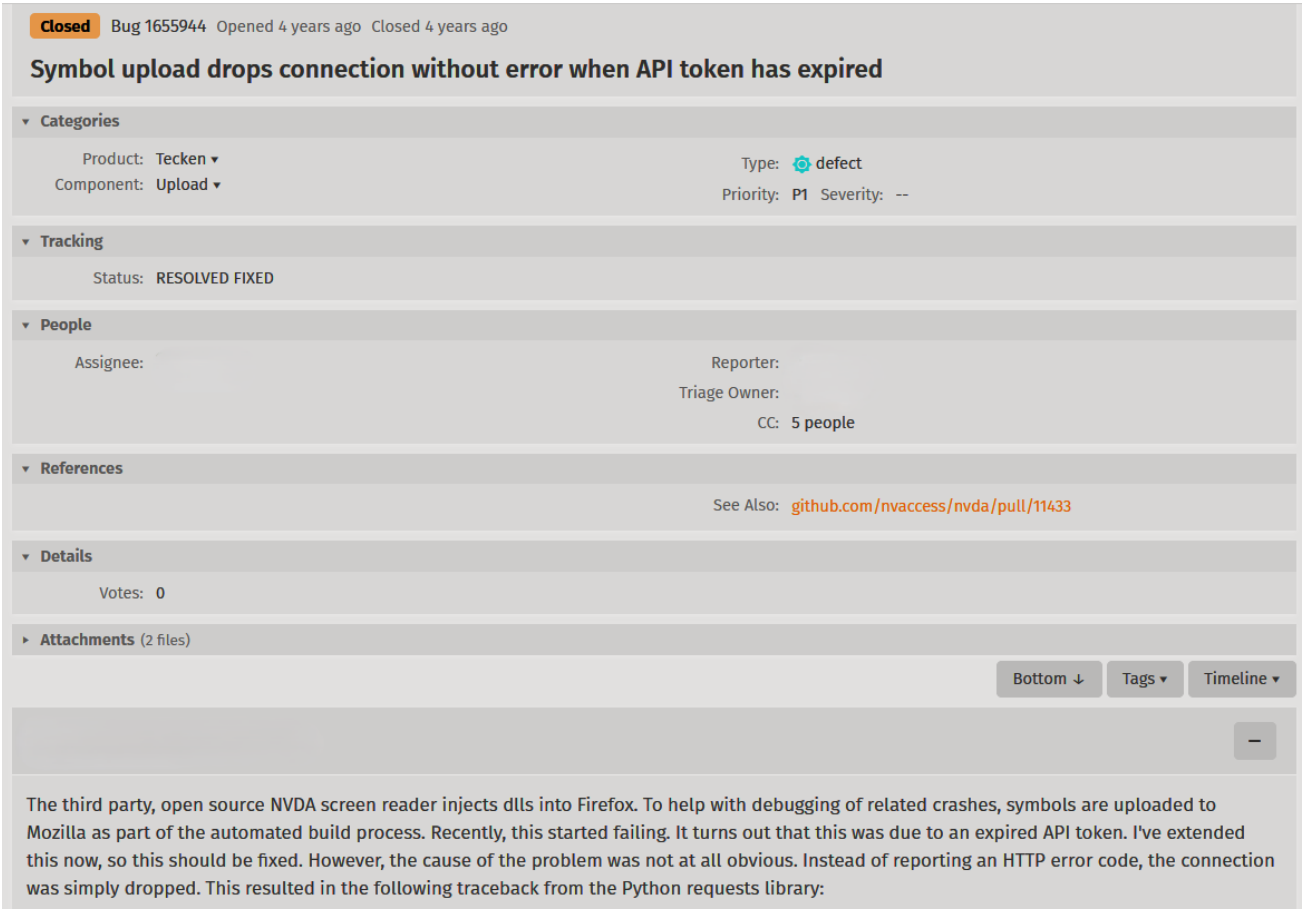


Figura 1: Relatório de Bug retirado do Bugzilla [13].

Alguns outros campos do relatório são adicionados após o relatório ser avaliado: a severidade, indicando a gravidade do problema e o nível de prioridade para a resolução; o responsável, que designa o desenvolvedor, ou equipe de desenvolvimento, que recebeu a função de corrigir o bug; e o campo de comentários que possibilita interações entre o relator do bug, o encarregado de investigar e resolver o problema e toda a comunidade de desenvolvimento. A Figura 1 mostra um exemplo real [13] de um relatório de bug que foi aberto na plataforma Bugzilla [7] e posteriormente resolvido.

Esse relatório descreve um problema relacionado a expiração de um token de API e como isso está afetando uma página. Na seção de categoria do relatório, a sinalização recebida foi de relatório que aponta um defeito, sendo esse de prioridade P1 e severidade não informada. A prioridade no Bugzilla existe em cinco classes (P1, P2, P3, P4 e P5), sendo a prioridade P1 a que possui um nível de prioridade para resolução mais alta, e a prioridade P5 a mais baixa. Na seção de monitoramento, o relatório foi

marcado como resolvido e fixado na plataforma. Na seção de pessoas, está a marcação do usuário redator do relatório e o usuário que assinou e resolveu o defeito apontado. As seções de referências e detalhes que apontam os links dos repositórios onde as soluções foram inseridas e se houve mais alguma adição ou contribuição da comunidade de desenvolvimento. A seção de arquivos mostra todos os arquivos que foram adicionados no relatório pelas seções de descrição e na de comentários. Por fim, a seção de comentários por ser complexa e extensa não foi adicionada na figura, mas está disponível para acesso público no link relacionado ao relatório de bug que foi disposto neste documento. Relatórios de bugs recebem classificações de status e resolução, um tipo de status possível é 'RESOLVED' que mostra à comunidade que o relatório de bug foi finalizado e o problema mensurado foi corrigido.

A análise dos campos estruturais de um relatório de bug, como *status*, prioridade e resolução, tem sido explorada na literatura como forma de compreender seu ciclo de vida e os fatores associados ao tempo e ao desfecho da resolução. Nesse contexto, estudos anteriores investigaram como essas informações são utilizadas ao longo do processo de triagem e resolução de defeitos. Ferreira J. [24] analisou a influência de campos de relatórios para determinar o status de resolução dos relatórios de bugs. O campo *status* concentra informações sobre a linha do tempo de mudanças durante a resolução do relatório de bug. Essas mudanças denotam avanços importantes, como, por exemplo, a atribuição de um desenvolvedor para analisar o problema, ou se o problema teve uma solução aplicada corrigindo-o e, portanto, se o relatório já pode ser finalizado.

Em tese, a principal função de um relatório de bug é descrever de forma clara e precisa o bug encontrado, incluindo informações sobre como reproduzi-lo, o ambiente em que ocorreu e a sua gravidade. Estudos buscam compreender e inferir quais são as principais características que possibilitam que um relatório de bug seja bem-sucedido ao detalhar e ser guia na resolução do problema apontado em um curto intervalo de tempo durante o ciclo de vida do relatório. Gromova et al. [3] conduziram um estudo acerca de relatórios de bugs de alto impacto, onde os autores analisaram manualmente relatórios e extraíram recursos que eram relatados. Como resultados, obtiveram que os campos e recursos de casos de teste, exemplos de código, etapas de reprodução e

rastreamento de pilha são fundamentais para os relatórios e comumente solicitados pelos desenvolvedores com solicitações adicionais. Ferreira J. [24] estudou como os campos textuais de um relatório de bug podem influenciar na resolução de relatórios. Como resultados, o autor obteve que os campos de resumo, descrição e comentários foram considerados importantes por contribuírem significativamente para relatórios com resolução final 'FIXED'. M. R. Karim [21] realizou um estudo exploratório utilizando relatórios de bugs de projetos open-source [4]. Como resultados, o autor também obteve achados que mostram que os recursos relacionados a campos de etapas de reprodução, casos de testes, exemplos de código, rastreamento de pilha e comportamento esperado são os recursos mais solicitados durante a resolução do relatório de bugs, sendo de grande importância para a correção rápida e eficiente do problema.

Nos estudos citados, seus objetivos e resultados sugerem que determinados campos dos relatórios de bugs exercem papel relevante durante o processo de resolução. Até onde foi possível verificar na literatura revisada neste trabalho, não foram encontrados estudos que tenham como foco principal a análise da importância da seção de comentários de um relatório de bug, em especial no que se refere à sua influência direta na resolução e no tempo necessário para a correção do problema.

1.2 Problema

A seção de comentários de um relatório de bugs constitui uma importante fonte de comunicação entre desenvolvedores e relatores de problemas. A troca contínua de informações nesse espaço, aliada à facilidade de interação entre os participantes, amplia a circulação de ideias na comunidade. A Figura 2 mostra um exemplo de comentário retirado de um relatório de bug [16]. Em tal, um desenvolvedor da comunidade discute e cobra opções melhores para a reprodução do problema mencionado, citando exemplo de outra experiência dele.

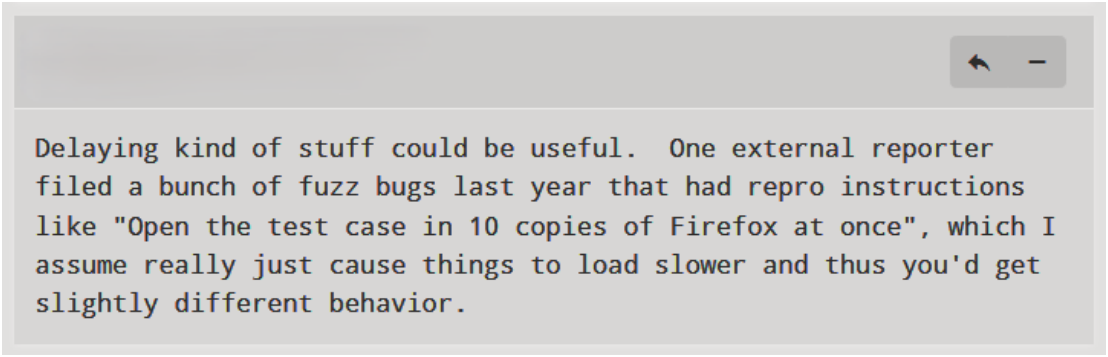


Figura 2: Comentário retirado do relatório de bug ID: 955888 [16].

Diante da relevância da seção de comentários como espaço de interação e colaboração, estudos têm buscado compreender de que forma essas interações contribuem para o processo de resolução dos relatórios de bugs. Nesse contexto, o estudo de S. Breu et al. [5] teve como foco a análise do impacto da seção de comentários em relatórios. Eles analisaram aspectos relacionados a perguntas feitas na seção de comentários de um relatório. Foram categorizadas as questões extraídas de uma porção de relatórios e divididas em grupos, demonstrando posteriormente o valor dessas perguntas, analisando quantitativa e qualitativamente sobre quando as perguntas são feitas e como são respondidas. Como resultados, eles obtiveram que o envolvimento constante do relator é fator determinante para relatórios bem-sucedidos. Embora importante e alinhado ao escopo, o estudo de S. Breu et al. [5] trouxe contribuições para compreender parte da dinâmica comunicacional presente na seção de comentários de relatórios de bugs, porém ele delimitou sua análise apenas a comentários com perguntas explícitas, desconsiderando outros comentários que podem ser relevantes para o progresso da resolução de relatórios de bugs. Assim, aspectos como sugestões técnicas, atualizações, soluções parciais, diretrizes e manifestações com conversas e dúvidas não foram considerados, o que limita a compreensão sobre o papel da seção de comentários dentro do processo colaborativo da resolução de problemas de software.

Nesse contexto, o presente estudo tem como foco a continuação do estudo de Ferreira J. [24] que evidenciou a importância de campos textuais para o sucesso na resolução de relatórios de bugs. Enquanto o estudo de Ferreira J. [24] aborda a influência geral dos campos, este trabalho se propõe a investigar de forma mais

profunda o impacto da seção de comentários, analisando seus diferentes tipos, funções comunicativas e contribuições para com o andamento e eficiência na detecção do tempo de vida de resolução de relatórios de bugs. Com isso, buscamos preencher uma lacuna ainda existente na literatura aplicando o entendimento sobre como as interações textuais entre desenvolvedores e relatores através dos comentários podem influenciar no tempo de vida dos relatórios de bugs.

1.3 Objetivos

Considerando o problema disposto na subseção anterior, o presente estudo propõe modelos para classificação de comentários de relatórios de bugs através do uso de algoritmos de machine learning em relatórios de bugs. Para isso, consideramos importante estabelecer um padrão na classificação dos comentários em tipos fortemente definidos. Dessa forma, buscamos classificar comentários, considerando também o contexto de anexos associados quando aplicável, e investigar como essa classificação pode estar relacionada à identificação de relatórios de bug com maiores probabilidades de serem resolvidos em tempos curtos ou longos. Para tal, os objetivos deste trabalho são:

1. a criação um método para classificação de comentários com base no seu conteúdo textuais e anexos que possam possuir;
2. a criação de modelos de classificação de comentários através de machine learning;
3. a criação de modelos de previsão da duração de resolução de relatórios de bugs em relatórios de longo ou curto prazo através de machine learning;
4. a condução de uma análise dos tipos de comentários mais presentes em relatórios de bugs com uso da tipagem oferecida no método de classificação criado;
5. a condução de uma análise da influência de comentários, suas classificações, e do histórico de modificações de relatórios de bugs para determinar como tais podem influenciar no tempo de resolução de um relatório de bug.

Com essas análises, esperamos que os modelos treinados sejam capazes de identificar se relatórios de bug irão ter um tempo de resolução de curto ou longo prazo através dos comentários que possuem em sua seção de comentários, os seus tipos classificados, e do histórico de modificações de cada relatório de bug.

1.4 Questões de Pesquisa

Para o presente estudo foram elaboradas três questões de pesquisa que focam especificamente nos objetivos almejados neste estudo:

QP1: Como os comentários de relatórios de bugs podem ser classificados seguindo seu conteúdo textual, anexos e influência no processo de resolução de um relatório de bug?

Essa questão busca investigar padrões recorrentes nos comentários de relatórios de bugs, considerando aspectos técnicos, comunicativos e colaborativos ao longo do processo de resolução. A análise resultante visa compreender como esses elementos podem ser sistematizados em uma estrutura categórica consistente, que posteriormente possa ser utilizada como base para experimentos de classificação automática.

QP2: De que forma as características e classificações dos comentários influenciam na identificação do tempo de resolução dos relatórios de bugs?

Essa questão busca investigar se existem associações entre os tipos de comentários e a duração do ciclo de vida dos relatórios, analisando se determinados padrões de interação estão relacionados a tempos de resolução mais rápidos ou mais prolongados.

QP3: Como as mudanças no histórico de modificações dos relatórios de bugs e os tipos de comentários em relatórios de bugs se relacionam com o tempo de resolução do relatório de bug?

Através dessa questão buscamos compreender a influência do histórico de modificações e dos comentários dos relatórios de bugs no tempo de resolução dos

relatórios, identificando se há padrões combinados entre a dinâmica do histórico e o conteúdo dos comentários que possam resultar em relatórios de bugs com tempos para resolução maior ou menores.

1.5 Escopo

Este estudo realiza uma análise de relatórios de bugs provenientes da plataforma Bugzilla, utilizados em projetos de software mantidos pela Fundação Mozilla. A investigação concentra-se na estrutura dos relatórios de bugs, com ênfase na seção de comentários, considerando sua relação com o processo de resolução dos relatórios e com o tempo necessário para a correção dos problemas reportados.

Embora o recorte empírico deste estudo esteja centrado em relatórios provenientes do Bugzilla em projetos mantidos pela Fundação Mozilla, a abordagem proposta não se restringe exclusivamente a comunidades open source. Os elementos analisados, como histórico de modificações, comentários, anexos e padrões iniciais de interação, também estão presentes em contextos corporativos e industriais que utilizam sistemas de rastreamento de defeitos. Assim, os métodos de classificação e análise investigados nesta dissertação podem, em princípio, ser adaptados para outros cenários de manutenção de software, desde que existam registros estruturados equivalentes.

O escopo desta pesquisa contempla relatórios de bugs registrados em um intervalo temporal recente e representativo, bem como seus respectivos comentários e históricos de modificações. Não fazem parte do escopo empírico deste estudo a análise de relatórios provenientes de outras plataformas de rastreamento de bugs ou a avaliação de métricas relacionadas ao código-fonte dos projetos analisados. Ainda assim, essa delimitação não impede que a abordagem metodológica proposta seja futuramente avaliada em outros ambientes de desenvolvimento que utilizem artefatos e registros semelhantes.

1.6 Contribuições

As contribuições reivindicadas com esse estudo incluem:

1. uma taxonomia para classificação de tipos de comentários e de tipos de anexos;
2. um conjunto de comentários classificados de forma manual utilizando o sistema de classes formulado;
3. um modelo de classificação de comentários utilizando algoritmos de machine learning que sejam capazes de identificar tipos de comentários;
4. uma análise correlacional entre os tipos dos comentários e o tempo de resolução dos relatórios de bugs;
5. um modelo de previsão da duração de resolução de relatórios de bugs utilizando algoritmos de machine learning que sejam capazes de identificar se relatórios de bugs serão rápidos ou lentos com base em comentários e mudanças de histórico;
6. uma análise entre o histórico de modificações dos relatórios e os tipos de comentários e em como eles podem influenciar no tempo de resolução do relatório de bug.

1.7 Estrutura do Documento

Os próximos capítulos deste estudo estão dispostos da seguinte forma: o capítulo 2 apresenta a fundamentação com definições gerais dos conceitos e do ferramental necessários para o melhor entendimento do estudo; o capítulo 3 apresenta os trabalhos relacionados com este estudo; o capítulo 4 dispõe do método de pesquisa e como ocorreu o desenvolvimento do estudo; o capítulo 5 mostra a taxonomia desenvolvida; o capítulo 6 mostra os resultados da classificação manual de comentários; os capítulos 7 e 8 mostram os resultados os experimentos de classificação de comentários e da previsão da duração da resolução de relatórios de bugs; e o capítulo 9 dispõe das conclusões e dos próximos passos do estudo assim como as considerações finais sobre o mesmo.

2 Fundamentação Teórica

Este capítulo tem como objetivo apresentar e discutir os conceitos teóricos e trabalhos relacionados que fundamentam o desenvolvimento deste estudo. São abordados os principais conceitos associados à plataforma Bugzilla, à estrutura e características de relatórios de bugs, bem como aos algoritmos de aprendizagem de máquina. Essa fundamentação fornece o embasamento necessário para a definição da metodologia adotada, a seleção dos dados analisados e a interpretação dos resultados obtidos.

2.1 Relatórios de Bugs

Relatórios de bugs são registros que descrevem detalhadamente bugs, falhas e problemas encontrados em softwares durante o processo de desenvolvimento, de teste ou uso do sistema. Esses relatórios podem assumir diversas formas, mas em geral incluem informações importantes sobre o software, o ambiente de execução em que o bug ocorreu, o processo de etapas para reproduzir o problema, e eventuais informações adicionais que sejam necessárias. Relatórios de bugs podem ser submetidos por qualquer pessoa que encontre o bug, incluindo usuários finais, testadores e desenvolvedores. Esses relatos têm alto grau de importância, sem eles os desenvolvedores teriam mais dificuldade para encontrar e corrigir falhas no software. Somado a isso, eles fornecem feedbacks importantes dos usuários de como o software está funcionando.

Além dos campos informacionais apresentados anteriormente, como o título do relatório, a descrição, os passos para reprodução, os anexos, a severidade, o desenvolvedor responsável e a seção de comentários, os relatórios de bugs passam, ao longo de seu ciclo de vida, por um processo de acompanhamento e finalização. Nesse processo, após a análise do problema relatado, os desenvolvedores pelo sistema preenchem campos adicionais que indicam o andamento e o desfecho do relatório. Dentre esses campos, destacam-se o *status*, que descreve a situação atual do relatório ao longo de seu ciclo de tratamento, e o campo de *resolution*, que indica a decisão final

tomada em relação ao problema reportado. A seguir, são descritas as possíveis classificações dos campos *status* e *resolution.*

O campo *status* informa como está o andamento do relatório. As possibilidades de *status* são UNCONFIRMED, NEW, ASSIGNED, REOPENED, RESOLVED, VERIFIED e CLOSED. O status UNCONFIRMED denota que o problema mencionado em relatório não teve reprodução por outros desenvolvedores; NEW denota um relatório recém aberto sem nenhuma atualização; ASSIGNED confirma que o relatório foi visto e atribuído a um desenvolvedor para resolvê-lo; REOPENED denota que um relatório que já tinha sido marcado como resolvido por outro desenvolvedor foi reaberto para análise; VERIFIED confirma que o relatório foi analisado e teve o problema descrito reproduzido em outras condições; CLOSED indica que o relatório teve seu problema resolvido e foi fechado; por fim RESOLVED indica que o relatório teve seu problema resolvido mas ainda está em aberto para que a comunidade de desenvolvimento possa realizar revisões e ou atualizações sobre.

O campo *resolution* demonstra como foi determinada a resolução final do relatório. As possibilidades são INVALID, WONTFIX, INACTIVE, DUPLICATE, WORKSFORME, INCOMPLETE, EXPIRED, MOVED e FIXED. Relatório de bug INVALID demonstra que o relatório foi considerado inválido pela equipe; WONTFIX denota um relatório que aponta para um bug real mas a equipe responsável não planeja corrigir por algum motivo; DUPLICATE marca o relatório de bug como um duplicata de outro relatório já existente para o mesmo; WORKSFORME demonstra que a equipe tentou e não conseguiu reproduzir o bug relatado; INCOMPLETE sendo uma variação para invalidar um relatório devido à pouca informação sobre o problema, impedindo a reprodução apropriada; EXPIRED indica que o relatório de bug ficou inativo a espera do autor adicionar mais informações sobre o problema, mas devido ao tempo excessivo ele foi finalizado; MOVED marcando que o relatório de bug foi movido para outro local não sendo o bugzilla o local apropriado para tal; e, por fim, a *resolution* FIXED, indica que o relatório de bug foi finalizado e corrigido com uma atualização de código específica para o problema.

Consideramos mostrar e detalhar as possibilidades para os campos de *status* e *resolution* de um relatório interligados exclusivamente ao Bugzilla, de onde foram

extraídos os dados para o estudo. Outras ferramentas especialistas em relatórios de bugs podem possuir outras nomenclaturas e possibilidades para tais campos.

2.2 Bugzilla

Bugzilla [7] é um sistema de código aberto utilizado comumente para documentar, rastrear bugs e problemas, sendo popular entre desenvolvedores de software. O Bugzilla oferece uma interface web que possibilita aos desenvolvedores submeterem, gerenciar e rastrear bugs e problemas que tenham sido identificados em softwares. A Figura 3 demonstra a interface do Bugzilla que permite inserir um novo relatório de bug.

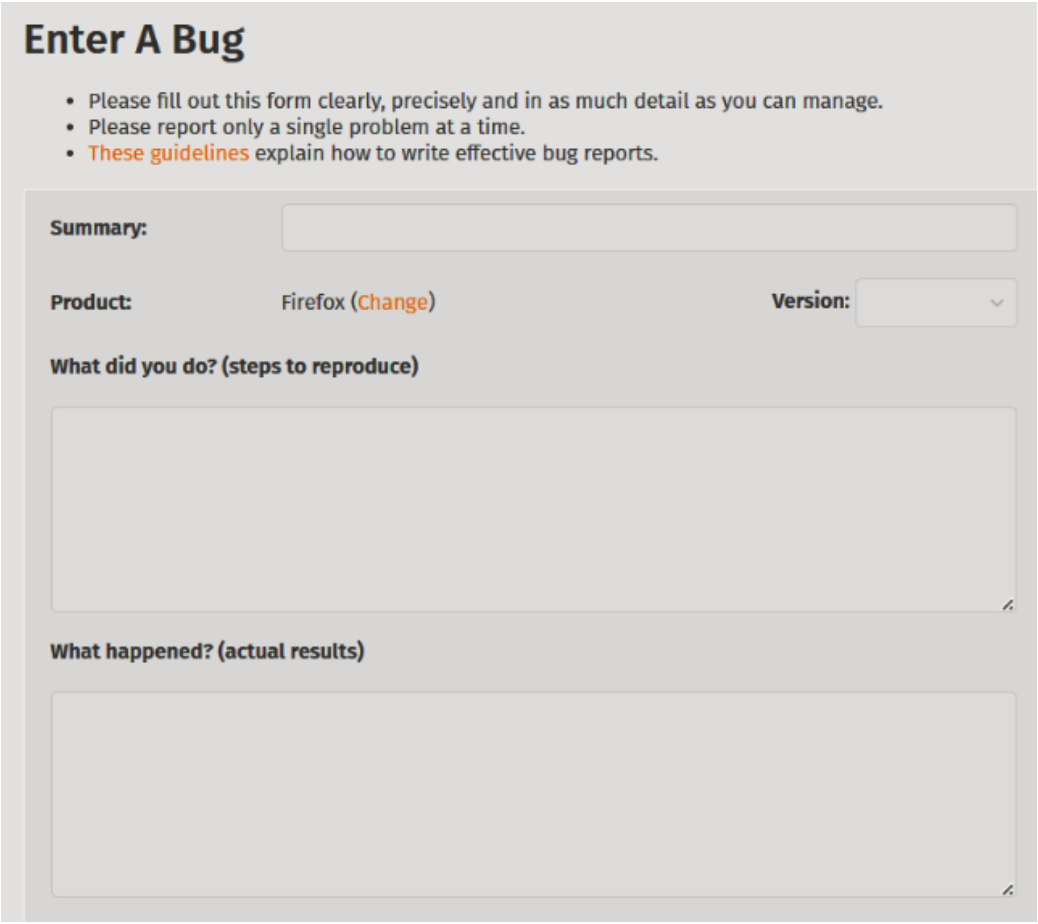


Figura 3: Página web para escrever novo relatório de Bug no Bugzilla [15]

Conforme a Figura 3, ao criar um relatório de bug, é necessário descrever inicialmente detalhes do problema, o campo *summary*, para fornecer uma descrição leve e sucinta do problema; o campo produto e versão, para identificar em qual produto e

sua respectiva versão ocorreu ou ocorre o problema; o campo de passos para reproduzir, onde relata-se a sequência de eventos que foram acionados no produto para ocorrer o erro; e o campo de resultado atual, para informar qual o comportamento atual que o problema gerou.

A Figura 4 mostra informações adicionais solicitadas para descrever em detalhes sobre o problema relatado, entre essas informações tem-se: o campo de resultados esperados, onde o autor do relatório deve informar qual era o comportamento esperado em contraste com o comportamento atual do erro; o campo de incluir arquivos, permitindo que sejam anexados imagens e arquivos para detalhar melhor o problema; o tipo de bug, indicando se o relatório está reportando um problema ou sugerindo uma melhoria para alguma funcionalidade; e por fim as flags de segurança e detalhes adicionais para anotar o relatório como sendo uma prioridade devido ao seu potencial risco de comprometer outros usuários e informações extras de qual plataforma de acesso ocorreu o problema.

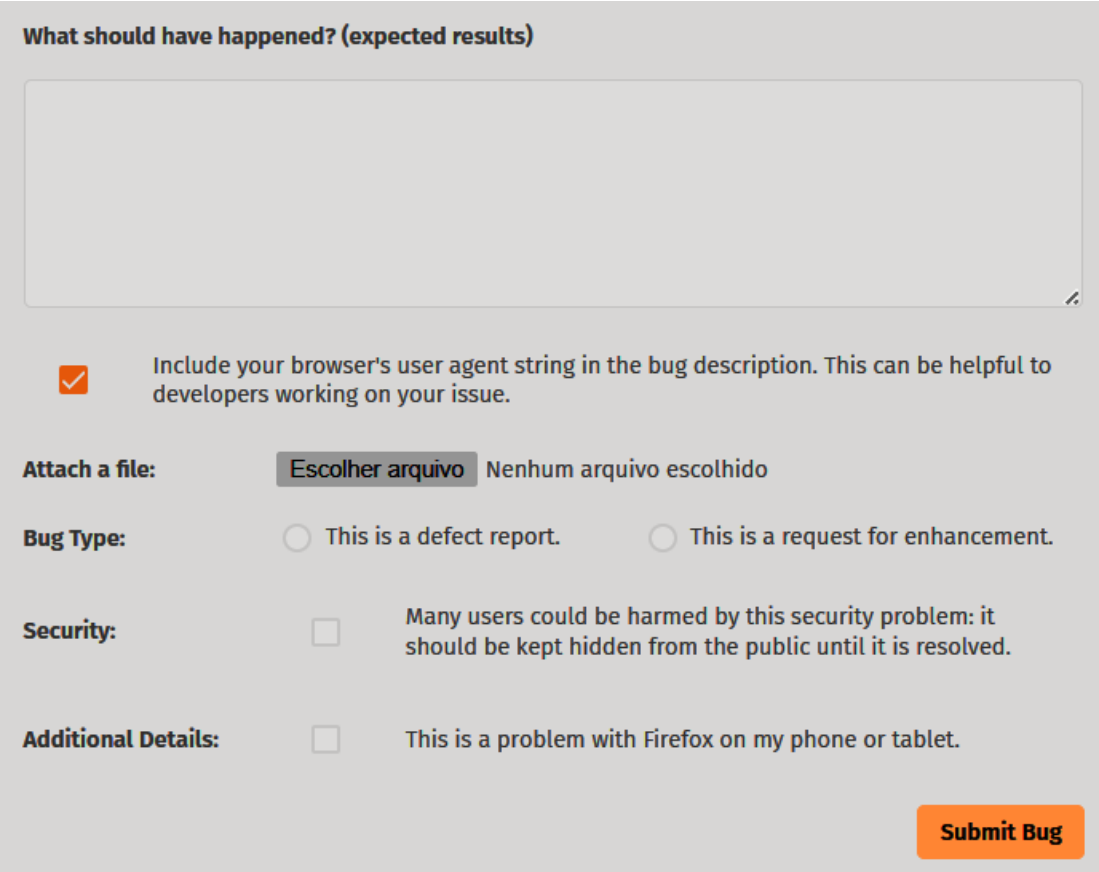


Figura 4: Página web para informações adicionais em novo relatório de Bug no Bugzilla

Com o relatório de bug preenchido, o autor consegue submetê-lo na plataforma do Bugzilla para posteriormente ser analisado, e caso constatado ser realmente um problema, ser resolvido pela comunidade de desenvolvimento. Organizações como a Fundação Mozilla [8], a Fundação Eclipse [9] e projetos de código aberto (*open source*) [4] têm adotado o Bugzilla [7] como ferramenta oficial para controle de bugs. Além de seu uso prático no desenvolvimento de software, o Bugzilla também é amplamente utilizado em estudos, servindo como base de dados para pesquisas que investigam a estrutura, o conteúdo e o ciclo de vida de relatórios de bugs, como nos trabalhos de Calixto et al. [23], Ferreira [24], Catolino et al. [6], Zhang et al. e Breu et al. [5].

A flexibilidade e capacidade de personalização do Bugzilla se integra com plataformas e ferramentas de desenvolvimento de software, sistemas de gerenciamento de código-fonte, de projetos e servidores. O Bugzilla facilitou a viabilidade do estudo por permitir através da sua interface de busca REST, requisições de conjuntos de

relatórios de bugs dentro do escopo do estudo para serem analisados. Os dados foram retirados e processados através do pipeline desenvolvido neste trabalho e armazenados em um *dataset* com todas as informações cruciais consideradas pelo estudo. Detalhes sobre o experimento e o *dataset* do experimento a ser executado neste trabalho são descritos no capítulo 4.

2.3 Machine Learning

Machine Learning [10] é um campo da inteligência artificial que opera com o objetivo de predizer informações e resultados através da manipulação e uso de dados com modelos e algoritmos específicos para identificar padrões e obter conclusões. Existem vários tipos de algoritmos de *Machine Learning*, cada um projetado para entender e lidar com diferentes contextos e dados fornecidos. Dentre eles, destacam-se, no contexto do desenvolvimento de software e da pesquisa científica, o Aprendizado Supervisionado, o Aprendizado não Supervisionado e os Modelos de Linguagem de Grande Escala (Large Language Models - LLMs), por serem amplamente empregados na análise de dados de engenharia de software. Esses métodos permitem desde a classificação automática de artefatos de software até a identificação de padrões e relações ocultas em grandes volumes de dados, sendo fundamentais em estudos.

No Aprendizado Supervisionado, o modelo é treinado com conjuntos de dados que foram rotulados, os exemplos de entrada estão associados às saídas esperadas. Dessa forma, o algoritmo entende o mapeamento desses dados permitindo que ele possa ser utilizado para prever ou classificar dados fora do conjunto original utilizado. Nesse estudo, o objetivo inicial é montar uma base de dados com os comentários sendo rotulados manualmente com auxílio da taxonomia desenvolvida. Com a base de dados, a próxima parte é analisar e treinar modelos de *machine learning* para classificar comentários de forma automática.

No Aprendizado Não Supervisionado, o modelo é treinado com dados não rotulados, sem que haja informações das saídas esperadas. O objetivo desse treinamento é encontrar estruturas de dados, padrões ou relações usando os próprios dados sem que haja nenhuma orientação externa durante o processo. O objetivo para com o uso desse

tipo de treinamento, neste trabalho, é encontrar e analisar padrões envolvendo o conteúdo textual dos comentários, firmando informações que são predominantes em comentários. N. Limsettho et al. [11] conduziram o desenvolvimento de um framework para categorizar relatórios de bugs de acordo com características e estruturas ocultas usando abordagens de aprendizagem de máquina não supervisionada por *clusterings*. Eles conseguiram agrupar automaticamente relatórios de bugs com semelhanças textuais, e obtiveram desempenhos comparáveis com abordagens de aprendizagem supervisionada.

LLMs (Large Language Models) [12], ou modelos de linguagem grandes, são softwares de inteligência artificial com grande capacidade para entender e gerar textos, dentre outros tipos de tarefas. Esses modelos são treinados com grandes quantidades de dados e utilizam técnicas de aprendizagem de máquina profunda para serem capazes de compreender, reconhecer e interpretar a linguagem humana ou outros tipos de dados complexos, que podem ser imagens, vídeos e arquivos de tipos variados. Apesar de serem uma tecnologia recente, as LLMs já são empregadas em atividades como classificação e sumarização de textos, análise de grandes volumes de dados, geração e apoio ao desenvolvimento de código-fonte, além de aplicações em pesquisas científicas voltadas à análise de relatórios de bugs.

3 Trabalhos Relacionados

Este capítulo foca na discussão da literatura relacionada sobre trabalhos com foco em relatórios de bugs que se assemelham a este trabalho e, portanto, seguem como guias de incentivo para o desenvolvimento do mesmo. Existe uma gama de pesquisas que focam na análise de campos e composições que são predominantes em relatórios de bugs.

Calixto F. et al. [23] estudou mudanças dentro de relatórios de bugs para entender quais campos sofrem mais mudanças e como elas podem se relacionar. O estudo mostrou que relatórios que demonstram bugs válidos têm os campos de prioridade e gravidade [média - alta] frequentemente modificados. No estudo foi utilizado um *dataset* composto por relatórios de bugs extraídos do Bugzilla, provenientes de projetos da Fundação Mozilla. Os autores observaram que relatórios considerados válidos apresentam modificações frequentes em campos como prioridade e severidade. Em contraste com o presente trabalho, que analisa o conteúdo textual da seção de comentários e o histórico de modificações do relatório de bug na influência no tempo de resolução, o trabalho de Calixto et al. concentra-se nas mudanças estruturais dos campos dos relatórios, sem explorar semanticamente a influência dos comentários e do histórico de modificações no tempo de resolução do relatório de bug.

T. D. Sasso et al. [25] realizaram análises sobre uma amostra de relatórios de bugs extraídos de bases de dados de projetos de software de código aberto, incluindo o Bugzilla. O *dataset* do estudo foi composto por relatórios contendo diferentes quantidades de campos preenchidos, e o objetivo foi investigar se o número de campos fornecidos influencia diretamente o tempo de resolução do bug. Os resultados indicaram que o aumento no número de campos preenchidos apresenta baixa correlação com o tempo de resolução. Diferentemente do presente estudo, os autores não realizaram análise dos comentários e do histórico de modificações, limitando-se a aspectos quantitativos e estruturais dos relatórios.

Ferreira J. [24] conduziu um estudo para classificar relatórios com campos de *status* e *resolution* diversos e identificar quais campos poderiam influenciar a conclusão de relatórios de bugs com resolução ‘FIXED’. O trabalho utilizou um dataset de

relatórios de bugs extraídos do Bugzilla envolvendo projetos de software de código aberto. Como resultados o autor destacou que campos textuais como resumo, descrição e comentários são os campos mais significativos para os relatórios classificados, assim como a adição de anexos em comentários durante a resolução dos relatórios de bugs. O estudo de Ferreira J. constitui um ponto de partida direto para o presente trabalho; contudo, enquanto Ferreira J. focou na classificação de relatórios, o presente estudo aprofunda-se na classificação dos comentários individualmente, analisando seus tipos, conteúdos textuais e relações com o tempo de resolução.

Xia Xin. et al. [32] propuseram modelos automatizados para apoiar a triagem de relatórios de bugs por meio de técnicas de aprendizagem de máquina. O *dataset* utilizado foi composto por relatórios históricos extraídos do Bugzilla de projetos de código aberto, incluindo Mozilla e NetBeans [26], considerando campos estruturados (*component* e *priority*) e campos textuais (*summary* e *description*) para prever a necessidade de reatribuição e/ou refinamento de tais campos. O estudo atingiu resultados como F1-Score acima de 0.8 mostrando que o conteúdo textual dos relatórios é uma das principais fontes de informação para melhorar a precisão de atribuições iniciais em relatórios de bugs. Este estudo é relevante pois demonstrou que o texto de campos livres dos relatórios, como os comentários, é valioso na identificação e triagem de relatórios de bugs, o que converge com o foco deste presente estudo que é explorar principalmente os comentários e as modificações ao longo do histórico de relatórios de bugs para determinar influências sobre o tempo de resolução dos mesmos.

Em geral, os trabalhos buscam encontrar os principais fatores responsáveis por facilitar e acelerar o processo de resolução de problemas. Com isso, analisamos trabalhos que se relacionam amplamente com este nos quesitos de classificação e análise das seções de comentários e o histórico de modificações de relatórios de bugs.

Catolino et. al [6] realizaram um estudo voltado ao desenvolvimento de uma taxonomia de tipos de relatórios de bugs, com o objetivo de compreender e caracterizar as diferentes naturezas dos bugs reportados. O *dataset* analisado foi composto por 1.280 relatórios de bugs extraídos de 119 projetos de software, pertencentes aos projetos Mozilla, Apache e Eclipse, que utilizam o Bugzilla como sistema de rastreamento. A partir da análise do conteúdo textual dos relatórios, os autores identificaram nove tipos

predominantes de relatórios de bugs, abrangendo diferentes categorias de defeitos, como problemas relacionados à interface do usuário, desempenho, configuração e outros aspectos técnicos do software. Com base nessa taxonomia, foi proposta uma abordagem automatizada para classificar relatórios de bugs, alcançando uma pontuação de 64% na medida F. Em contraste, o presente estudo não tem como foco a classificação de relatórios como um todo, mas sim a classificação e análise dos comentários individualmente, investigando como diferentes tipos de comentários, seus conteúdos textuais e o histórico de modificações podem influenciar no tempo de resolução dos relatórios.

Zhang et al. [33] conduziram um estudo empírico sobre os fatores que podem impactar no tempo de resolução de bugs em projetos relacionados ao Eclipse, utilizando um *dataset* de relatórios de bugs extraídos do Bugzilla. Este estudo separou o processo de resolução em cinco intervalos temporais, desde o relato inicial até a resolução final, e analisou métricas referentes ao conteúdo dos relatórios de bugs, as propriedades do código e as mudanças realizadas durante a resolução. Por meio de testes estatísticos e modelos de regressão logística, os autores demonstraram que os maiores atrasos em relatórios ocorrem antes e depois de modificações de código, sendo influenciados por fatores como severidade, complexidade do código e extensão dos comentários. Este estudo se relaciona diretamente com o presente trabalho por evidenciar a relevância da comunicação textual e de interações através de comentários como elementos determinantes no tempo total de resolução de relatórios de bugs. Contudo, o estudo não analisou o texto dos comentários, concentrando-se em características quantitativas e estruturais dos campos textuais de relatórios de bugs, mostrando diferenças com este presente estudo que propõe analisar a seção de comentários como um todo para inferir as influências sobre os tipos de comentários, e seus conteúdos textuais, no tempo de resolução de um relatório de bug.

S. Breu et al. [5] realizaram um estudo com foco na análise de perguntas presentes em relatórios de bugs. O *dataset* foi composto por 600 relatórios de bugs mantidos no Bugzilla e relacionados aos projetos Mozilla e Eclipse, dos quais extraíram um total de 947 questões atribuídas nos relatórios. Através do desenvolvimento da pesquisa, foi identificado oito categorias específicas de questões levantadas, sendo elas

divididas em um total de 40 grupos. Dessas questões foram analisados aspectos como o tipo de pergunta, sendo a maioria relacionadas à depuração e correção do bug relatado, e aspectos relacionados à taxa e tempo necessário para uma resposta à pergunta. Como resultados, os autores concluíram que perguntas em relatórios de bugs, se respondidas de forma rápida, contribuem para o avanço do status e conclusão do relatório. Do ponto de vista taxonômico, o trabalho de Breu et al. propõe uma categorização voltada especificamente para comentários interrogativos, estruturando perguntas em categorias e grupos que refletem suas intenções (por exemplo, depuração e correção). Em contraste, a taxonomia proposta nesta dissertação amplia esse recorte ao abranger diferentes funções comunicativas e técnicas observadas na seção de comentários como um todo, incluindo atualizações, diretrizes, soluções parciais, descrições técnicas e manifestações automatizadas, além de considerar tipos de anexos associados. Assim, enquanto Breu et al. focam na classificação de perguntas e seus efeitos sobre a resposta, o presente estudo busca representar a diversidade de papéis dos comentários e investigar sua relação com o tempo de resolução dos relatórios.

4 Metodologia

Este capítulo descreve o método de pesquisa utilizado no estudo assim como as etapas de execução do experimento como a obtenção dos dados, os tratamentos e procedimentos com os comentários, as etapas de classificação, experimentação com modelos de machine learning e as questões de validação. Os subcapítulos deste capítulo foram organizados de modo a refletir o fluxo metodológico apresentado na Figura 5. Os subcapítulos referentes à coleta de dados, ao processamento dos comentários, à definição das classes de comentários e aos experimentos de classificação correspondem diretamente às fases ilustradas na Figura 5. As demais subseções descrevem procedimentos auxiliares, variações experimentais e aspectos de avaliação e validação que dão suporte às etapas principais do estudo, mas que não são representados como fases independentes no fluxo metodológico.

4.1 Visão Geral

Nosso objetivo é investigar relações existentes entre a seção de comentários de relatórios de bugs e seu histórico de modificações em detrimento com o tempo necessário para com sua resolução. Propomos uma sequência de passos para atingirmos esse objetivo. Acreditamos que a investigação e a proposição de modelos e técnicas de *Machine Learning* capazes de classificar comentários de relatórios de bugs permitem identificar informações relevantes sobre os tipos e as qualidades desses comentários. Tais informações podem ser relacionadas ao tempo de resolução dos problemas descritos nos relatórios de bugs. Para tal, desenvolvemos um estudo em etapas que busca obter dados reais de relatórios de bugs e comentários, tratar e normalizar tais dados de comentários, modelar um sistema de classificação para comentários de relatórios de bugs, utilizar modelos e técnicas de machine learning para classificação automatizada de comentários de relatórios de bugs e por fim analisar e coletar os resultados.

Uma etapa de caracterização descritiva dos dados, por si só, não seria suficiente para responder integralmente às questões de pesquisa formuladas neste trabalho. A

caracterização permite identificar distribuições, frequências e padrões gerais dos comentários e dos relatórios, mas não possibilita avaliar, de forma sistemática, a capacidade de generalização desses padrões nem seu poder preditivo sobre o tempo de resolução. Por essa razão, o estudo foi estruturado em múltiplas frentes complementares, sendo elas: a classificação supervisionada que foi empregada para verificar se os tipos de comentários definidos na taxonomia podem ser reconhecidos automaticamente; o agrupamento não supervisionado que foi utilizado como etapa exploratória para investigar se estruturas naturais emergem dos dados sem rótulos prévios; e os modelos de regressão e classificação temporal que foram adotados para analisar em que medida características textuais e estruturais permitem prever a duração da resolução dos relatórios de bugs. Dessa forma, a combinação entre caracterização, classificação, agrupamento e regressão permitiu avançar da descrição dos dados para a análise explicativa e preditiva proposta nas questões de pesquisa.

A Figura 5 representa visualmente todas as etapas deste presente estudo, na qual é possível visualizar as atividades que foram realizadas e os artefatos que foram produzidos e como se mostram fundamentais para outras etapas. De forma geral, a Figura 5 ilustra o fluxo metodológico adotado neste estudo, iniciando pela coleta de dados no Bugzilla, seguida pela extração e processamento dos comentários e modificações dos relatórios. Em seguida, são apresentadas as etapas de definição de classes de comentários e a rotulação manual de comentários com uso dessas classes de comentários, a preparação dos conjuntos de dados para treino e teste e a execução dos experimentos de classificação de comentários e de relatórios de bugs. Por fim, o fluxo contempla a análise dos resultados obtidos a partir dos modelos aplicados, encerrando o processo metodológico do estudo.

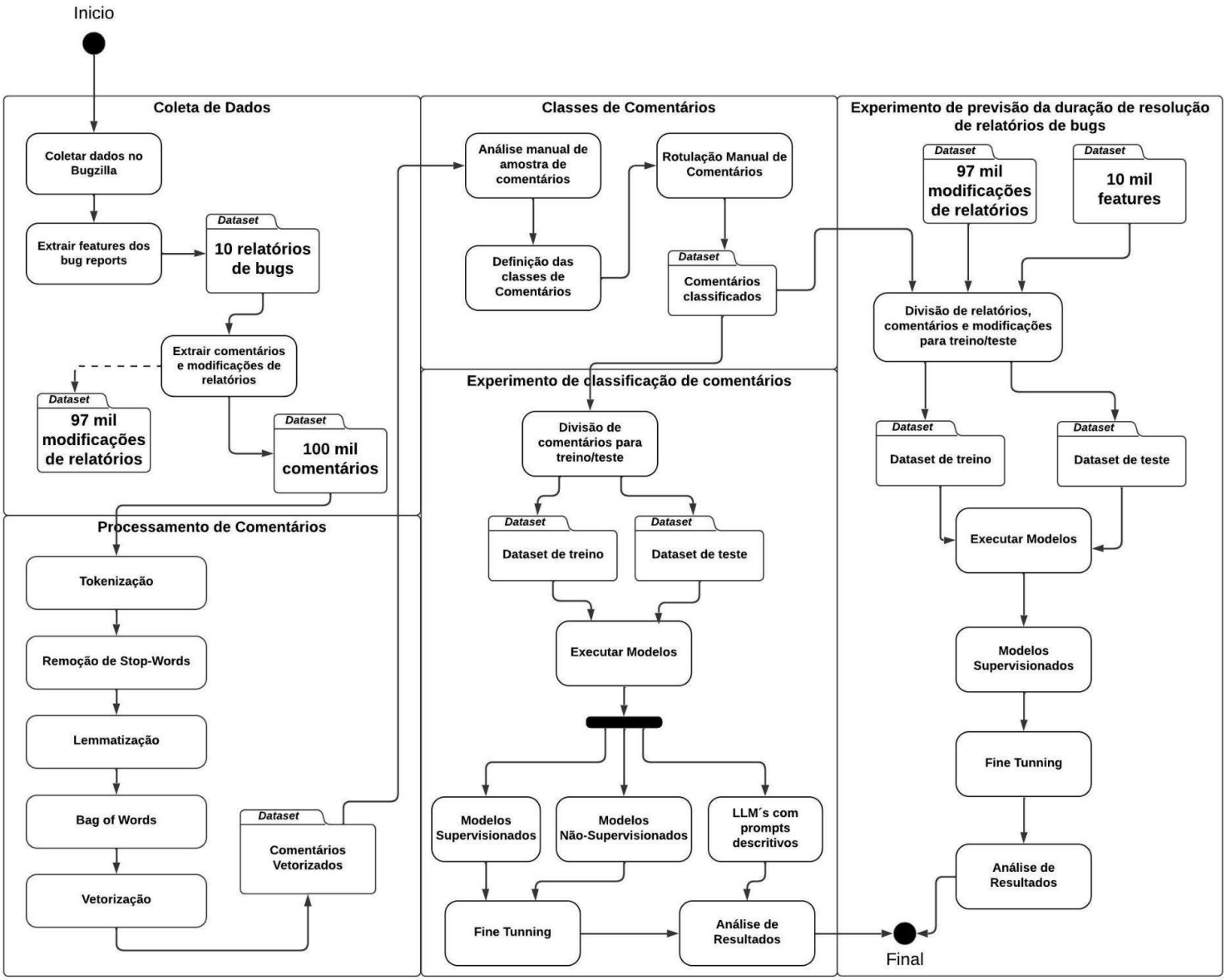


Figura 5: Visão geral do fluxo metodológico do estudo, apresentando as etapas de coleta de dados, processamento dos comentários, definição das classes, experimentos de classificação e análise dos resultados.

4.2 Coleta de dados

A Coleta de Dados é a primeira etapa do estudo. O processo de obtenção dos dados de relatórios de bugs foi realizado por meio de acessos à instância do Bugzilla mantida

pela Fundação Mozilla, que hospeda relatórios de bugs de projetos de software de código aberto. Para tal, foram desenvolvidos scripts com o objetivo de automatizar a obtenção de todas as informações necessárias de um conjunto de relatórios de bugs. Esses scripts encontram-se disponíveis em uma pasta compartilhada no link [34]. Nessa etapa, foram necessárias duas sub-etapa de obtenção de dados: a sub-etapa de obtenção de features de relatórios de bugs; e a sub-etapa de obtenção de comentários e histórico de modificações dos relatórios de bugs, que são descritas a seguir.

4.2.1 Obter Features dos relatórios de bugs

O script *Data Requests* [34] inicialmente obtém as informações básicas através de uma requisição ao Bugzilla que solicita a ele informações de dez mil relatórios de bugs no Bugzilla. Consideramos informações básicas as informações de id, status, resolução, data de criação, data da última modificação e o componente ao qual o relatório de bug tem referência. Estas informações são armazenadas em um *dataset* específico. Nesta sub-etapa de obtenção de features de relatórios de bugs, devido a restrições da própria plataforma do Bugzilla, foi fixado o valor de dez mil relatórios de bugs para que a requisição não ocasionasse erro ou resposta mal formatada. O modelo de dados gerado nessa etapa para representar os relatórios de bugs é representado na Figura 6. Esse modelo segue a representação informacional de JSON e consequentemente é salvo no formato de dataset em CSV. Como features necessárias para esse estudo, consideramos as seguintes informações dos relatórios de bugs:

- *bug_id* sendo o id único do relatório de bug dentro da plataforma do Bugzilla, sendo essa informação de extrema importância por permitir que outras informações como os comentários do relatório possam ser buscadas e indexadas com base nesse identificador;
- *cf_last_resolved* sendo a data de última modificação ou mudança de status que foi registrada para o relatório de bug;
- *status* sendo o status atual que o relatório de bug se encontra na plataforma;
- *creation_time* sendo a informação da data que o relatório de bug foi criado;

- *product* sendo a referência do produto ou serviço do qual o relatório de bug foi gerado dentro da plataforma do Bugzilla;
- *resolution* sendo a resolução final que foi atribuída ao relatório de bug.

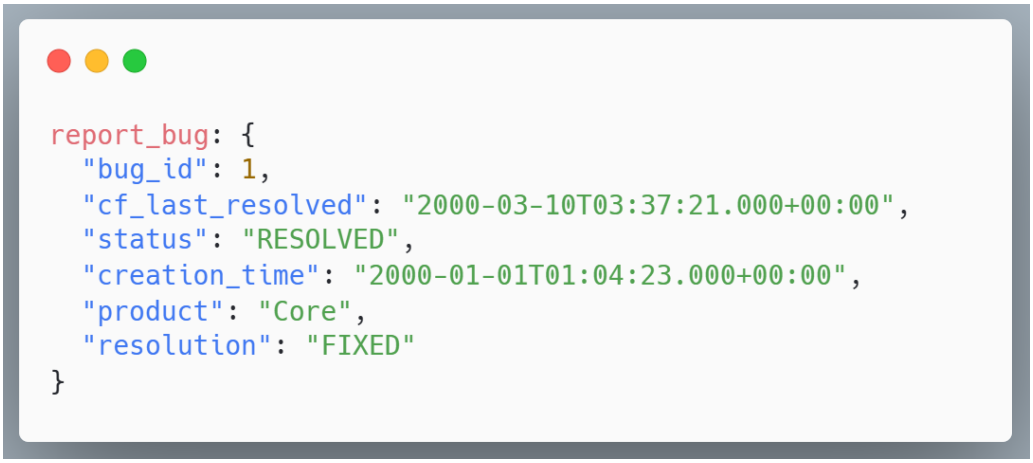


Figura 6: Representação do modelo de dados de um relatório de bug

Os relatórios analisados não pertencem exclusivamente a um único projeto. A base contempla registros distribuídos entre diferentes projetos identificados pelo campo *product*. A distribuição quantitativa por projeto foi analisada por meio do notebook Quantitative Analyses, disponível nos scripts do estudo [34], permitindo verificar a heterogeneidade da amostra. Essa heterogeneidade é relevante, pois diferentes projetos podem apresentar níveis distintos de maturidade, modelos de governança e dinâmicas comunitárias que impactam a forma como os bugs são tratados.

4.2.2 Obter comentários e histórico de modificações

Nesta sub-etapa de obtenção dos comentários e histórico de modificações dos relatórios de bugs, é utilizado o mesmo script de *Data Request* [34] empregado na etapa anterior, agora configurado para realizar uma iteração sobre os identificadores *bug_id* de cada relatório previamente requisitado ao Bugzilla. Nesta sub-etapa, novas requisições são realizadas, passando como parâmetros o identificador de cada relatório e os campos específicos a serem obtidos: os comentários e o histórico de modificações de cada um

dos relatórios incluídos no *dataset* de relatórios de bugs. Após a obtenção dessas informações, o script realiza a formatação dos dados e os adiciona em *datasets* específicos, sendo um destinado aos comentários e outro ao histórico de modificações. Esses dados são indexados diretamente pelos identificadores dos relatórios aos quais pertencem.

Os comentários retornados pela API são organizados pelo Bugzilla em ordem cronológica dentro de cada relatório de bug. Embora a plataforma não adote uma estrutura hierárquica explícita de encadeamento entre respostas, a sequência temporal permite reconstruir a evolução da discussão ao longo do ciclo de vida do relatório. Essa sequência foi considerada durante a etapa de classificação manual, permitindo interpretar o contexto conversacional da thread. Para fins de modelagem, entretanto, a unidade de análise permaneceu sendo o comentário individual.

O modelo de dados gerado nessa etapa para representar os comentários é representado na Figura 7. Esse modelo segue a representação informacional de JSON e consequentemente é salvo no formato de *dataset* em CSV. Como features necessárias para esse estudo, consideramos as seguintes informações dos comentários como relevantes:

- *id_bug* esse identificador é adicionado em cada comentário para indexar o comentário ao seu relatório de bug de origem;
- *id_comment* sendo o identificador único do comentário na plataforma Bugzilla;
- *creation_time* sendo a data que o comentário foi criado;
- *count* sendo o identificador do comentário dentro da lista de comentários que o relatório de bug possui;
- *raw_text* sendo o conteúdo textual original do comentário;
- *text* sendo o conteúdo textual do comentário com formatações da própria plataforma do Bugzilla;
- *creator* sendo a informação do usuário que criou o comentário;
- *attachment_id* correspondendo ao id relativo à possibilidade de o comentário possuir um anexo vinculado ao mesmo;
- *is_private* sendo uma flag de identificação se o comentário é privado na plataforma do Bugzilla.



Figura 7: Representação do modelo de dados de um comentário

Além da captura dos comentários, essa etapa também captura o histórico de modificações que ocorrem em cada relatório de bug. O histórico de modificações refere-se às alterações estruturais registradas no relatório, tais como mudanças de status, prioridade, resolução e outros campos administrativos. Embora a plataforma registre eventos relacionados à publicação de comentários, o conteúdo textual dos comentários não foi tratado como entidade sujeita a versionamento neste estudo. A análise textual considerou a versão disponível no momento da extração dos dados. O modelo de dados gerado nessa etapa para representar o histórico de modificações é representado na Figura 8. Como features necessárias para esse estudo, consideramos as seguintes informações do histórico de modificações:

- *id_bug* o identificador único do relatório de bug que é adicionado em cada histórico de modificação para permitir uma indexação com o relatório de bug de origem;
- *author* sendo o autor que realizou as modificações no relatório de bug;
- *time* sendo a data da modificação;

- *changes* sendo uma lista das modificações realizadas, onde cada modificação possui os campos:
    - *field_name* sendo a modificação;
    - *added* sendo o que foi adicionado;
    - *removed* o que foi removido no relatório durante a modificação.



Figura 8: Representação do modelo de dados de modificações de histórico

A identificação do autor foi preservada no dataset, permitindo distinguir os diferentes participantes da discussão em cada relatório. Essa informação foi considerada na análise contextual durante a etapa de classificação manual, ainda que não tenha sido utilizada como variável independente nos experimentos de classificação textual.

4.3 Caracterização dos Dados

A composição dos dados utilizados neste estudo é dividida em três tipos principais: relatórios de bugs, comentários de relatórios de bugs e histórico de modificações dos relatórios de bugs. Os dados foram extraídos da plataforma Bugzilla e referem-se a projetos de software internos e externos mantidos pela Fundação Mozilla.

Os relatórios obtidos não pertencem exclusivamente a um único projeto. A base contempla registros distribuídos entre diferentes projetos identificados pelo campo

product. A distribuição quantitativa por projeto foi analisada por meio do notebook *Quantitative Analyses*, disponível nos scripts do estudo [34], permitindo verificar a heterogeneidade da amostra. Essa heterogeneidade é relevante, pois diferentes projetos podem apresentar níveis distintos de maturidade, modelos de governança e dinâmicas comunitárias que impactam a forma como os bugs são tratados.

Os relatórios de bugs constituem uma amostra de 10.000 (dez mil) objetos, selecionados de acordo com critérios específicos para garantir consistência, diversidade e relevância dos dados analisados. Foram considerados relatórios:

1. Com data de abertura entre 2014 e 2024;
2. Que possuam um status de *resolution* como 'FIXED', 'WORKSFORME', 'DUPLICATE', 'WONTFIX', 'INVALID', 'INCOMPLETE', 'INACTIVE', 'MOVED' e 'NEW';
3. Que possuam uma seção de comentários com pelo menos 2 (dois) comentários;

Optou-se por não restringir a análise apenas a relatórios com resolução final 'FIXED'. A inclusão de diferentes tipos de desfecho, bem como de relatórios ainda em aberto, permite evitar viés de sobrevivência e capturar de forma mais realista a dinâmica do ciclo de vida dos relatórios. A distribuição das resoluções foi analisada no notebook *Quantitative Analyses* [34], garantindo transparência na caracterização da base utilizada.

Os comentários compõem uma amostra de 110.106 (cento e dez mil cento e seis) registros, correspondentes aos relatórios de bugs selecionados. Já o histórico de modificações reúne aproximadamente 97.000 (noventa e sete mil) registros, representando as alterações realizadas ao longo do ciclo de vida dos relatórios. Para viabilizar a manipulação e análise dos dados, foram considerados apenas features dos relatórios de bugs que apresentassem impacto direto para o estudo. Entre as features utilizadas estão:

1. ID **(id):** o número de identificação único do relatório de bug;
2. Status **(status):** o status final do relatório de bug;
3. Última mudança **(cf_last_resolved)**: a data da última modificação feita no relatório pelo relator ou desenvolvedor responsável;

4. Comentários **(comments)**: à seção de comentários do relatório de bug;
5. Histórico **(history)**: o registro de todas as mudanças feitas durante o tempo de resolução do relatório.

A aplicação desses filtros teve como objetivo reduzir a complexidade dos dados retornados pelo Bugzilla, tornando o processamento mais eficiente e direcionado aos objetivos da pesquisa.

4.4 Processamento de Comentários

A etapa de Processamento de Comentários é a segunda etapa do estudo, ela é possível após o *dataset* de comentários ter sido finalizado na etapa anterior de coleta de dados. O objetivo dessa etapa é preparar o conteúdo textual de todos os comentários para as próximas etapas do estudo. As próximas subseções detalham o objetivo, as técnicas utilizadas e os resultados do processamento dos comentários.

4.4.1 Processamento de Texto

Nessa etapa são realizadas o processamento, formatação, contagem de palavras chaves, tokenização, lematização e armazenamento dessas informações no próprio *dataset* de comentários. A primeira atividade é processar e formatar o comentário, sendo feitas verificações para identificar se o comentário tem algumas informações, e por meio desse processo, novas features são adicionadas no modelo de dados representacional de comentários que anteriormente foi demonstrado na Figura 7. Dentre as novas features adicionadas, temos como: *is_reply* que identifica se o comentário é uma resposta a outro comentário; *attachment_created* para identificar se o comentário está adicionando um anexo no relatório; *attachment_comment* para identificar se o comentário é direcionado diretamente a uma análise de algum anexo; e por fim, *has_code_block* para identificar se o comentário contém algum trecho de código.

Em seguida são realizadas as atividades de Tokenização e Lematização: a tokenização consiste em separar o texto retirando espaços e o transformando em uma

lista de tokens; a lematização atua na lista de tokens, retirando pontuações e atuando em cada palavra reduzindo-a em seu lema que é a forma base da palavra; A atividade de contagem de palavras identifica, no conteúdo tokenizado e lemmatizado, as bases de palavras chaves que possam estar presentes usando como referência um dicionário montado no estudo que contém uma variedade de palavras no contexto de computação. Ao identificar essas palavras no texto, a contagem de tais palavras é adicionada em uma nova feature de nome *keyword_count*. Além das etapas de processamento textual, esta fase também contempla a representação vetorial dos comentários, conforme ilustrado na Figura 5 de visão geral do estudo.

Após a normalização do texto, os comentários são convertidos para uma representação baseada em *Bag of Words* [35], na qual cada documento é representado por um vetor cuja dimensionalidade corresponde ao vocabulário extraído do corpo do comentário. Sobre essa representação, é aplicada a ponderação TF-IDF (*Term Frequency–Inverse Document Frequency*), que ajusta o peso de cada termo de acordo com sua frequência no documento e sua distribuição no conjunto total de textos, reduzindo a influência de termos excessivamente frequentes e pouco discriminativos. A dimensionalidade do espaço vetorial é determinada pelo número de termos únicos presentes no vocabulário após as etapas de tokenização, lematização e filtragem textual. Não foram aplicadas técnicas adicionais de redução de dimensionalidade, como Análise de Componentes Principais (PCA) ou métodos explícitos de seleção de atributos. A ordem de grandeza da dimensionalidade situa-se na faixa de milhares de atributos, valor típico em tarefas de classificação textual baseadas em representações vetoriais esparsas. Por fim, essas informações e o texto processado são inseridos no *dataset* de comentários.

O texto formatado, tokenizado e lemmatizado é adicionado na feature de *full_formatted_text*, que guarda o texto com redução de ruídos presentes no conteúdo original, uma vez que, nas etapas anteriores, são aplicadas regras de formatação e filtragem que removem ou desconsideram trechos de código, links, caracteres especiais e outras estruturas não textuais que não contribuem para a análise linguística dos comentários. A Figura 9 representa o modelo final de dados dos comentários ao final da etapa de processamento de texto.



Figura 9: *Representação do modelo de dados de um comentário após processamento do texto*

4.5 Classes de Comentários

Esta etapa de classes de comentários consistiu na realização de uma análise manual dos comentários conduzida pelos pesquisadores responsáveis por este estudo, com o objetivo de identificar padrões e semelhanças textuais entre os comentários e, a partir disso, propor um conjunto de classes que abrangesse os diferentes tipos de comentários presentes no conjunto de dados. O processo foi iniciado a partir da seleção de uma amostra inicial de aproximadamente 950 comentários do *dataset*. Esses comentários foram analisados individualmente por meio de leitura qualitativa, buscando identificar a intenção comunicativa, o conteúdo técnico e o papel desempenhado pelo comentário no contexto do relatório de bug. Com base nessa análise, foram propostas classes preliminares, às quais os comentários eram associados conforme suas características predominantes.

À medida que novas classes eram definidas, os comentários previamente classificados passaram por revisões sucessivas, com o objetivo de verificar a adequação da classe atribuída e garantir a consistência do processo de categorização. A classificação manual foi conduzida pelo autor desta dissertação. Não foi realizada dupla anotação independente do corpus, motivo pelo qual não foi possível calcular medidas formais de concordância interavaliadora, como o coeficiente Kappa de Cohen. Entretanto, ao longo do processo de definição e refinamento das classes, os critérios adotados e os casos considerados ambíguos foram discutidos no âmbito das reuniões de acompanhamento de orientação acadêmica, buscando alinhar a interpretação conceitual das categorias propostas.

Nessas reuniões, exemplos representativos e casos ambíguos eram apresentados e discutidos até que se alcançasse consenso sobre a classe mais adequada. Esse alinhamento foi conduzido de forma qualitativa, sem medição quantitativa de divergências ou conflitos de rotulação. Esse procedimento colaborativo contribuiu para reduzir ambiguidades e reforçar a consistência interna da taxonomia, ainda que não tenha sido conduzida uma validação estatística formal de concordância.

A unidade de análise adotada neste estudo foi o comentário individual. Os anexos associados aos relatórios de bug não foram tratados como instâncias independentes de classificação. Quando presentes, os anexos foram considerados no contexto do comentário ao qual estavam vinculados, podendo influenciar a atribuição da classe quando continham informações técnicas relevantes, como trechos de código, logs ou evidências de reprodução do erro. Dessa forma, a categorização permaneceu centrada no conteúdo textual do comentário, ainda que o material anexado pudesse complementar a interpretação sem constituir uma entidade classificada separadamente.

Esse procedimento iterativo resultou na definição de um conjunto final de 10 (dez) classes de comentários, em que cada classe representa um grupo de comentários com características comuns. A descrição detalhada das classes definidas, bem como os critérios utilizados no processo de classificação manual, é apresentada no capítulo 5 de Taxonomia deste estudo.

Essa etapa é fundamental para a execução do experimento, pois estabelece a base conceitual e prática da classificação adotada no estudo. A partir da rotulação manual

dos comentários, torna-se possível conduzir as etapas subsequentes com maior consistência metodológica, uma vez que os modelos de classificação passam a ser treinados e avaliados com dados previamente rotulados. O processo de classificação manual dos comentários foi apoiado por um script auxiliar, de nome *Comment Classifier* [34], desenvolvido com o objetivo de organizar e apresentar os comentários de cada relatório de bug em ordem sequencial. Esse script não realiza a classificação automaticamente, mas facilita o trabalho do autor ao exibir o contexto completo das interações entre os comentários, permitindo uma análise mais consistente e informada no momento da atribuição das classes. Com isso, ao final dessa etapa foram classificados de forma manual o quantitativo de 37 (trinta e sete) mil comentários do *dataset* de 110 (cento e dez) mil comentários.

4.6 Experimento de classificação de comentários

Esta etapa é subdividida em diversas sub-etapas com o objetivo de investigar e propor modelos e técnicas de machine learning que sejam capazes de classificar comentários de relatórios de bugs. Para tal, selecionamos modelos de machine learning que possuem técnicas de treinamento com supervisão, técnicas sem supervisão, modelos de linguagem de grande escala (LLMs), tecnologias de variações de dados para serem utilizadas nos *datasets* de dados, e ajustes de hiperparâmetros para possibilitar a execução de diversos cenários combinados com tais modelos gerando uma janela de possibilidades para encontrar os melhores modelos para classificação de comentários de relatórios de bugs.

Essa etapa foi desenvolvida e executada em ambientes do *Google Colab* [36], com o uso de bibliotecas consolidadas da linguagem Python. Para a implementação e avaliação dos modelos de *Machine Learning* foram utilizadas principalmente as bibliotecas *scikit-learn* [37], para os modelos de classificação supervisionada (*Naive Bayes* [38], *Support Vector Machines* - SVM [39], *Random Forest* [40], *Logistic Regression* [41] e *Decision Tree* [51]) e para os algoritmos de agrupamento *K-Means* [42] e *DBSCAN* [43]; *TensorFlow* [44] e *Keras* [45], para a implementação do modelo baseado em Redes Neurais Convolucionais (CNN) [46] ; e a biblioteca *HDBSCAN* [47],

empregada para o algoritmo de agrupamento homônimo. Além disso, foram utilizadas as bibliotecas *Pandas* [48] e *Numpy* [49] para manipulação e processamento dos dados, bem como bibliotecas de processamento de linguagem natural para suporte às etapas de preparação textual descritas anteriormente. Como fluxo principal do experimento, sendo este independente do modelo de machine learning usado para a classificação de comentários, temos inicialmente a importação e preparação dos dados que advêm das etapas anteriores desse estudo. Esses dados são extraídos de um arquivo CSV contendo os comentários pré -processados. Cada um dos registros deste arquivo representa um comentário associado a um relatório de bug, acompanhado de seu respectivo rótulo ("*type_comment*"), que define a classe que foi manualmente atribuída pelo autor anteriormente. Em seguida, os dados são divididos em conjuntos de treinamento e teste mantendo a proporção de 80% dos dados para treino e 20% para teste, permitindo, dessa forma, uma avaliação objetiva dos modelos sobre dados não vistos em treinos.

Para a representação do conteúdo textual dos comentários, foi adotado o modelo de vetorização TF-IDF (Term Frequency-Inverse Document Frequency) [50]. Essa técnica transforma o texto em uma matriz numérica que pondera a importância relativa de cada termo em relação ao corpo total do texto, permitindo assim que o modelo aprenda padrões linguísticos relevantes ao invés de realizar, por exemplo, apenas uma contagem de palavras.

Ao finalizar a etapa comum de importação e preparação dos comentários e seus conteúdos textuais, o fluxo de execução de experimento de classificação de comentários segue para a parte de execução e avaliação dos cenários para com os modelos. Por cenário, consideramos a seguinte junção: um modelo de machine learning dentre os escolhidos (seja ele supervisionado, não supervisionado, ou baseado em LLM); uma variação do *dataset* de comentários (incluindo a versão desbalanceada e as versões obtidas a partir das técnicas de balanceamento descritas no subcapítulo 4.7 de Variações de Dados); e as combinações de hiperparâmetros ajustáveis dos modelos, detalhadas no subcapítulo 4.9 de Ajustes de Hiperparâmetros. A listagem completa dos cenários avaliados, bem como a descrição das técnicas empregadas em cada um deles, é apresentada nas seções subsequentes deste capítulo. As etapas seguintes descrevem os modelos e técnicas escolhidas.

O presente experimento está diretamente associado à QP1, cujo objetivo é investigar como os comentários de relatórios de bugs podem ser classificados considerando seu conteúdo textual, a presença de anexos e seu papel no processo de resolução. A partir da definição da taxonomia proposta e da rotulação manual dos comentários, esta etapa busca avaliar empiricamente a viabilidade da classificação automática desses comentários por meio de algoritmos de machine learning. Assim, os resultados apresentados nesta seção fornecem evidências concretas para responder à QP1, examinando tanto o desempenho dos modelos quanto os padrões de confusão entre classes.

4.6.1 Modelos Supervisionados

Modelos supervisionados são modelos de machine learning que possuem uma técnica de treinamento que utiliza bases de dados que possuem rótulos de identificação para entender e replicar padrões dos próprios dados. No contexto deste estudo, consideramos os comentários de relatórios de bugs como nossa principal base de dados. A classificação desenvolvida nas etapas anteriores atribui rótulos de identificação a uma margem quantitativa dos dados permitindo assim que fossem utilizados modelos com técnicas de aprendizado supervisionado.

No contexto deste estudo, os modelos supervisionados foram utilizados para duas tarefas distintas: a primeira sendo a classificação multiclasse dos comentários de acordo com a taxonomia definida na Seção 4.5; e a segunda sendo a classificação relacionada à previsão do tempo de resolução dos relatórios de bugs (curto ou longo), conforme descrito na Seção 4.8. Na primeira tarefa, a saída esperada consiste na atribuição de uma das classes definidas na taxonomia aos comentários. Na segunda tarefa, a saída corresponde à predição da categoria temporal associada ao ciclo de vida do relatório.

A escolha dos modelos supervisionados considerou critérios amplamente adotados na literatura, tais como a recorrência de uso em estudos relacionados à classificação de texto e engenharia de software; a capacidade de lidar com dados textuais de alta dimensionalidade; a diversidade de paradigmas de aprendizagem, de

modo a permitir comparações entre modelos lineares, probabilísticos, baseados em árvores e redes neurais; e a viabilidade computacional para experimentação em conjuntos de dados de grande porte. Com base nesses critérios, foram selecionados os seguintes modelos supervisionados para o estudo:

- *CNN* (Rede Neural Convolucional) [46] sendo muito usado para análise de imagens e texto, pois consegue identificar padrões visuais em partes distintas e depois juntar tudo;
- *Naive Bayes* [38] que aponta como um modelo mais simples e parte de análise de probabilidades, funcionando bem quando as características são independentes entre si;
- *Support Vector Machines* - SVM (Máquina de Vetores de Suporte) [39] que funciona tentando traçar uma linha (ou um limite) que separe bem as categorias, mesmo que elas estejam em lugares difíceis no espaço dos dados;
- *Decision Tre*e [51] que funciona como um fluxograma, onde cada decisão leva a um novo caminho, até chegar a uma conclusão;
- *Random Forest* [40] que funciona como um conjunto de várias árvores de decisão, o que ajuda a melhorar a precisão e reduzir erros;
- *Logistic Regression* [41] que é um modelo que tenta prever uma categoria usando uma fórmula que transforma os resultados em probabilidades.

4.6.2 Modelos Não Supervisionados

Modelos não supervisionados são modelos de machine learning onde as técnicas de treinamento independem de dados que possuem rótulos. Esses modelos são extremamente úteis em processos de agrupamentos de dados em clusters, as características principais dos dados são aguçadas nesses grupos.

No escopo deste estudo, os modelos não supervisionados foram utilizados com o objetivo de identificar padrões estruturais e agrupamentos naturais entre os comentários, sem o uso prévio de rótulos. A saída destes modelos consiste na atribuição de cada comentário a um cluster, permitindo analisar possíveis correspondências entre agrupamentos automáticos e as classes definidas manualmente.

A escolha dos modelos não supervisionados baseou-se na ampla adoção na literatura para tarefas de agrupamento de texto, a capacidade de lidar com dados de alta dimensionalidade e a complementaridade entre abordagens baseadas em centroides e em densidade. Os modelos selecionados foram:

- *K-means* [42] um dos modelos mais conhecidos nessa área, ele atua tentando dividir os dados em um número fixo de grupos, colocando cada ponto perto de um centro;
- *DBSCAN* [43] que atua não precisando saber quantos grupos existem e consegue encontrar grupos de tamanhos e formas diferentes, ignorando pontos isolados que não se encaixam bem (ruídos);
- *HDBSCAN* [47] que é uma versão melhorada do DBSCAN, porém que consegue lidar melhor com dados mais complexos, também organiza os grupos de forma hierárquica e consegue escolher automaticamente quantos grupos fazem mais sentido.

4.6.3 LLMs (Large Language Models)

LLMs (Large Language Models) [12] são modelos de inteligência artificial treinados com grandes quantidades de texto com o intuito de entender e gerar informações textuais. As LLMs são capazes de responder perguntas, completar frases, traduzir textos, entre outras tarefas relacionadas ao processamento de linguagem natural. A escolha das LLMs neste estudo foi motivada por: crescente adoção em pesquisas recentes; a capacidade de generalização sem a necessidade de treinamento extensivo específico; e potencial de capturar nuances semânticas difíceis de serem modeladas por abordagens tradicionais.

No presente estudo, as LLMs foram utilizadas como abordagem complementar para classificação textual, sendo instruídas por meio de prompts estruturados que descrevem a taxonomia definida manualmente. Não foi empregado mecanismo de Retrieval-Augmented Generation (RAG), uma vez que as instruções e definições das classes foram fornecidas diretamente no contexto do prompt. A interação com os modelos seguiu uma estratégia de prompt estruturado, na qual cada comentário era

apresentado juntamente com a descrição das classes possíveis, solicitando-se a atribuição da categoria mais adequada. As respostas foram posteriormente analisadas e comparadas com os rótulos previamente definidos. As LLMs selecionadas foram:

- *Gemini* [52] modelo recente do Google que combina várias técnicas avançadas para entender tanto texto quanto imagens, buscando ser mais versátil e preciso;
- T5 (Text-to-Text Transfer Transformer) [53] modelo capaz de transformar qualquer tarefa de linguagem em um problema de completar texto, o que o torna bastante flexível para diferentes aplicações;

4.7 Variações de Dados

Os dados utilizados neste estudo vêm das etapas anteriores de coleta de dados e são compostos por relatórios de bugs, comentários e modificações de histórico de relatórios de bugs. Em especial, consideramos os comentários como a fonte de dados primária que permite a análise e a obtenção dos objetivos que almejamos para este estudo. Como estamos lidando com dados reais, encontramos desbalanceamento entre as classes de comentários que foram classificados manualmente para as etapas de treinamento dos modelos de machine learning. Como forma de mitigar esse desbalanceamento, aplicamos diferentes técnicas de rebalanceamento de dados, com o objetivo de gerar variações do *dataset* que permitissem uma avaliação mais justa do desempenho dos modelos de *Machine Learning*. Esse desbalanceamento entre classes pode prejudicar o processo de aprendizado dos modelos, uma vez que estes tendem a favorecer as classes majoritárias em detrimento das classes com menor representatividade. Para lidar com esse problema, foram adotadas diferentes estratégias de rebalanceamento dos dados. Para mitigar isso, adicionamos as estratégias de rebalanceamento a seguir:

- *Oversampling* [54] técnica que aumenta a quantidade de exemplos da classe menor, copiando ou criando exemplos parecidos com os que já existem;
- *Undersampling* [54] técnica que realiza o processo contrário, diminuindo a quantidade de exemplos da classe maior para equilibrar;
- *SMOTE* [14] é uma técnica de oversampling mais avançada, ela cria exemplos sintéticos da classe menor ao invés de só duplicar os exemplos existentes;

- *Cluster Centroids* [54] é uma forma de undersampling onde os exemplos da classe maior são substituídos por pontos centrais (médias) de grupos, tentando manter a diversidade com menos dados;
- *Class Weights* [54] é uma forma de dizer ao modelo, durante o treinamento, que ele deve dar mais importância aos erros da classe menor, sem mudar o conjunto de dados. Essas técnicas ajudam a garantir que o modelo aprenda de forma mais justa e consiga identificar corretamente todas as classes, mesmo as que têm poucos exemplos, gerando assim cenários com modelos de machine learning que entendem e se adaptam melhor ao treinamento.

O processo de divisão dos dados em conjuntos de treinamento (80%) e teste (20%), conforme descrito na subseção 4.6, foi realizado antes da aplicação das técnicas de balanceamento. O balanceamento foi aplicado exclusivamente ao conjunto de treinamento, mantendo o conjunto de teste com a distribuição original das classes. Essa estratégia foi adotada com o objetivo de evitar vazamento de informação (*data leakage*) e preservar uma avaliação realista do desempenho dos modelos, permitindo comparar o impacto do balanceamento sobre o aprendizado sem comprometer a validade da etapa de teste.

4.8 Experimento de Previsão da Duração de Resolução de Relatórios de bugs

Esta etapa do estudo tem como objetivo de investigar e propor modelos e técnicas de machine learning que sejam capazes de classificar relatórios de bugs como sendo relatórios de bugs de rápida ou lenta resolução usando como base features do próprio relatório, como também a presença de comentários classificados deste relatório, seus anexos, e o histórico de mudanças presente.

O experimento foi conduzido e executado em ambientes do Google Colab, com o uso de bibliotecas consolidadas da linguagem Python. O pipeline desenvolvido segue um fluxo dividido em etapas principais que se iniciam na preparação dos dados até a avaliação final dos resultados dos modelos supervisionados utilizados nesse

experimento. Para este experimento, foram utilizados exclusivamente modelos supervisionados, os mesmos empregados na etapa de classificação de comentários. Modelos não supervisionados e LLMs não foram considerados nesta fase, uma vez que o objetivo é avaliar a capacidade preditiva dos modelos a partir de variáveis previamente definidas e rotuladas. A escolha pelos modelos supervisionados decorre dos resultados obtidos na etapa de classificação de comentários, cujos desempenhos são discutidos na seção 7 de Resultados do Experimento de classificação de comentários.

Embora seja tecnicamente possível solicitar a uma LLM uma estimativa direta sobre o tempo de resolução de um relatório com base em seu conteúdo textual, essa abordagem não foi adotada neste experimento. O objetivo desta etapa foi avaliar a capacidade preditiva de modelos supervisionados treinados especificamente sobre o dataset estruturado do estudo, utilizando variáveis quantitativas e categóricas extraídas de forma controlada.

A utilização de LLMs para estimativas diretas envolveria um paradigma distinto, baseado em inferência generativa e potencialmente dependente de contexto externo não controlado. Dessa forma, optou-se por manter a consistência metodológica e a comparabilidade entre modelos treinados sob as mesmas condições experimentais. A investigação do uso de LLMs para previsão direta de tempo de resolução configura uma oportunidade para trabalhos futuros.

Nesse experimento, os dados foram carregados de três fontes complementares, sendo elas: o *dataset* de relatórios de bugs que contém identificadores, datas de criação e resolução do relatório, status e a resolução final dos mesmos; o *dataset* de comentários que contêm todos os comentários desses relatórios, os anexos que possuem, e as classificações que cada comentário recebeu na fase de classificação; por fim, do *dataset* de histórico de mudanças, que contêm todas as alterações realizadas em campos dos relatórios ao longo do tempo de duração do mesmo. Esses dados foram integrados através do identificador único do bug, *"bug_id"*, assegurando assim a correspondência entre os diferentes conjuntos de dados. Após finalizar essa etapa, foi calculado o tempo total de resolução em dias do relatório *"time_to_resolve_days"* considerando a diferença entre a data de criação *"creation_time"* e a data de resolução *"cf_last_resolved"*.

Para evitar vazamento de informações, entendido como o uso de dados que só estariam disponíveis após a resolução do relatório durante o treinamento do modelo, foi aplicada uma técnica de janela temporal denominada *Early Window* [55], essa técnica limita as features utilizadas no experimento apenas para as features com data nos primeiros dias após a abertura do bug. Essa janela, definida como sendo em 14 dias, filtra tanto os comentários quanto às mudanças realizadas nesse período inicial, garantindo que o modelo aprenda apenas com informações disponíveis nos estágios iniciais do ciclo de vida do relatório. A partir desta janela temporal foram extraídas features numéricas e categóricas que descrevem o comportamento inicial dos relatórios. Essas features incluem: a contagem de comentários; a diversidade de tipos de comentários classificados, que reflete a variedade de interações textuais ocorridas; o número e natureza das modificações de campo, representando alterações em atributos como status e responsável; a frequência de alterações no histórico, que indica a dinâmica de evolução do relatório; e os indicadores derivados dos anexos e da atividade dos usuários, como a presença de arquivos anexados e a participação de diferentes autores.

Considerando também que as threads de comentários de um relatório de bug pode se estender ao longo de todo o seu ciclo de vida, a abordagem proposta não depende do encerramento completo da discussão para aplicação do modelo. A previsão foi estruturada para utilizar exclusivamente informações disponíveis em estágios iniciais do relatório, tais como metadados, primeiros comentários e modificações iniciais de histórico.

Dessa forma, o modelo pode ser aplicado progressivamente, à medida que novos comentários são adicionados, permitindo atualizações incrementais da estimativa sem a necessidade de aguardar o término da thread. Essa definição evita o uso de informações futuras e preserva a validade experimental da tarefa de previsão. Essas variáveis foram combinadas com atributos próprios dos relatórios como status, resolução e produto, resultando assim em um conjunto de features que caracteriza tanto a estrutura do bug, quanto a dinâmica inicial de sua resolução.

Para avaliar a contribuição relativa dos diferentes grupos de variáveis, foram conduzidas análises de importância de features durante a etapa de modelagem,

permitindo identificar o peso associado às variáveis derivadas dos comentários em comparação com atributos estruturais do relatório. É importante destacar que o modelo identifica padrões estatísticos de associação e não relações causais diretas. Dessa forma, embora seja possível observar a influência relativa de determinados grupos de atributos, não se afirma que a predição decorra exclusivamente das características dos comentários, mas sim da combinação das variáveis consideradas no experimento.

Os relatórios de bugs foram rotulados automaticamente com base no tempo de resolução, calculado a partir dos dados temporais disponíveis. Finalizada a etapa de preparação, segue-se para as etapas de treinamento e à comparação de diferentes modelos supervisionados na classificação. Nessa etapa do estudo houve a avaliação de desempenho foi conduzida sob duas abordagens complementares. A primeira, baseada em regressão, teve como objetivo avaliar a capacidade dos modelos em prever o tempo contínuo de resolução dos relatórios de bugs. A segunda, baseada em classificação, buscou identificar se um relatório tende a ser resolvido de forma rápida ou lenta, permitindo comparar o desempenho dos modelos em dois cenários de decisão: um cenário binário, com duas classes temporais; e outro cenário com quatro classes temporais mais distintas, ambos os cenários alinhados a um dos objetivos centrais do estudo.

Durante a modelagem dessa etapa do estudo, foram realizados ajustes de hiperparâmetros e avaliações de desempenho dos modelos supervisionados. Os procedimentos adotados, bem como as métricas utilizadas e a análise de importância das features, são detalhados nas subseções seguintes deste capítulo.

4.9 Ajustes de Hiperparâmetros

Ajustes de hiperparâmetros consiste numa etapa importante para melhorar o desempenho dos modelos, tanto nos supervisionados e não supervisionados quanto nos LLMs. Essas técnicas envolvem escolher os melhores valores para configurações que afetam o funcionamento dos modelos. Esses ajustes não são aprendidos durante o treinamento e precisam ser testados manualmente ou com a ajuda de algoritmos para encontrar a melhor combinação que traga bons resultados para o contexto do estudo.

Para isso, o ajuste de hiperparâmetros foi conduzido de forma sistemática, explorando diferentes combinações de valores possíveis para cada configuração relevante dos modelos supervisionados, não supervisionados e de linguagem de grande escala (LLMs). Esses ajustes incluem, entre outros, a definição de taxas de aprendizado, limiares de decisão, profundidades de estruturas hierárquicas, e parâmetros de agrupamento e separação de dados. Cada hiperparâmetro impacta diretamente a forma como os modelos se adaptam às características dos conjuntos de dados e a complexidade das classes a serem aprendidas.

Para automatizar e otimizar esse processo, foram utilizados métodos de busca sistemática que avaliam o desempenho de múltiplas combinações de valores e selecionam aquelas que produzem os melhores resultados conforme as métricas de avaliação definidas. O método adotado foi o *GridSearch* [56], uma abordagem amplamente utilizada que executa uma busca exaustiva em uma grade (*grid*) de parâmetros pré-definidos. Em cada iteração, o modelo é treinado e validado com diferentes combinações de valores, e o desempenho é avaliado por meio de validação cruzada (*cross-validation*), garantindo maior confiabilidade estatística dos resultados e reduzindo o risco de viés. Esses processos ajudam os modelos a se adaptarem melhor aos dados e a melhorar a precisão das previsões e agrupamentos.

4.10 Avaliação e Teste

Para realizar análises e comparações entre os resultados obtidos através dos modelos de classificação automáticos, estimamos usar métricas de acurácia, F1 Score, Revocação e Precisão para avaliar o desempenho dos modelos de classificação supervisionados que foram descritos nas seções anteriores. Para medir o desempenho dos modelos não supervisionados, estimamos utilizar o Coeficiente de Silhueta e o ARI (Coeficiente Rand Index Ajustado). Para medir o desempenho das tarefas de regressão usando modelos supervisionados, estimamos usar as métricas MAE (Erro Absoluto Médio), RMSE (Erro Quadrático Médio da Raiz) e R^2 (Coeficiente de Determinação).

Acurácia [27] é uma métrica utilizada para medir a performance geral de um modelo de machine learning em tarefas de classificação, indicando a proporção de

previsões corretas em relação ao total de previsões realizadas. Essa métrica corresponde à razão entre o número de classificações corretas, isto é, a soma de Positivos Verdadeiros (PV) e Negativos Verdadeiros (NV), e o total de instâncias avaliadas. Formalmente, pode ser representada pela seguinte equação:

$$Acurácia = \frac{PV + NV}{PV + NV + FP + FN}$$

onde:

- PV (Positivos Verdadeiros): instâncias positivas corretamente classificadas;
- NV (Negativos Verdadeiros): instâncias negativas corretamente classificadas;
- FP (Falsos Positivos): instâncias negativas incorretamente classificadas como positivas;
- FN (Falsos Negativos): instâncias positivas incorretamente classificadas como negativas.

A precisão [28] é uma métrica que mede o total de vezes em que o modelo de machine learning acerta uma classificação em relação ao total de vezes que ele tentou acertar. Em suma, a precisão pode ser calculada pelo número de Positivos Verdadeiros (sendo eles acertos verdadeiros) dividido pelo número de Positivos Verdadeiros somado ao número de Falsos Positivos (sendo um acerto errôneo), essa métrica é representada pela seguinte equação:

$$Precisão = PV \,/\, (PV + FP)$$

Revocação [28] é uma métrica que mede o total de vezes que o modelo de machine learning acerta uma classificação em relação ao total de vezes em que ele deveria ter acertado na classificação. A Revocação é calculada pelo número de Positivos Verdadeiros (acertos verdadeiros) dividido pelo número de Positivos Verdadeiros somado ao número de Negativos Falsos (erro ao invés de acerto), essa métrica pode ser representada pela seguinte equação:

$$Revocação = PV \,/\, (PV + FN)$$

O F1 Score [28] é uma métrica calculada através da combinação equilibrada entre a precisão e a Revocação (recall) de um modelo. O F1 Score é calculado sobre 2 (duas) vezes o valor da Precisão do modelo multiplicado pelo valor de Revocação do modelo, dividido pela Precisão do modelo somado a Revocação do modelo, essa métrica pode ser representada pela seguinte equação:

$$F1 = 2 \times (Precisão \times Revocação) / (Precisão + Revocação)$$

O coeficiente de Silhueta [29] é uma métrica para medir o nível de coesão dos dados contidos em um cluster [30], e como estes clusters são separados uns dos outros, essa métrica assume valores entre -1 e 1. Um cluster é um conjunto agrupado de dados similares entre si.

Valores do coeficiente de Silhueta próximos de 1 mostram que os elementos classificados estão bem agrupados dentro de seus respectivos clusters e são bem separados dos demais. Enquanto isso, valores próximos a 0 indicam sobreposições de grupos de elementos, e valores negativos indicam que um elemento está, em média, mais próximo dos elementos de outro cluster do que daqueles do cluster ao qual foi atribuído, sugerindo uma alocação inadequada durante o processo de agrupamento. No contexto deste estudo, a métrica de coeficiente de Silhueta será usada para avaliar a consistência e a qualidade dos clusters gerados pelos modelos não supervisionados.

O coeficiente Rand Index Ajustado (ARI) [31] é uma métrica utilizada para analisar clusters e avaliar a qualidade do agrupamento medindo a similaridade do cluster ajustado pelo modelo de classificação supervisionada em relação a um cluster verdadeiro usado como referência. Essa métrica também varia entre -1 e 1, sendo que valores mais próximos de 1 indicam uma alta concordância entre os agrupamentos gerados pelos modelos não supervisionados e os agrupamentos verdadeiros, usando como base a identificação de elementos em comum entre os clusters gerados e verdadeiros. O ARI será utilizado neste estudo para comparar os clusters automáticos com as classificações manuais da fase de classificação de comentários, verificando assim a capacidade dos algoritmos não supervisionados de reproduzir a estrutura semântica identificada através de avaliação humana.

O Erro Absoluto Médio (MAE) [57] é uma métrica que calcula a média das diferenças absolutas entre os valores preditos e os valores reais, ela mede o erro médio cometido pelo modelo em unidades da variável analisada, que para o caso desse estudo, é o tempo de dias para a resolução do relatório de bug. Essa métrica é útil já que fornece uma interpretação direta, quanto menor o valor de MAE, mais precisas são as previsões realizadas pelos modelos. A métrica é representada pela equação:

$$MAE = (1/n) \times \Sigma |yi - \hat{y}i|,$$

onde yi representa o valor real, ŷi o valor predito e n o número total de observações.

O Erro Quadrático Médio da Raiz (*Root Mean Squared Error – RMSE*) [58] é uma métrica amplamente utilizada para avaliar o desempenho de modelos preditivos, pois quantifica a magnitude média dos erros entre os valores estimados pelo modelo e os valores reais observados. Essa métrica considera o quadrado das diferenças entre valores reais e preditos, o que resulta em maior penalização para erros de grande magnitude, tornando o RMSE sensível a desvios significativos nas previsões. Além disso, por aplicar a raiz quadrada ao valor médio dos erros, o RMSE é expresso na mesma unidade da variável analisada, o que facilita sua interpretação e comparação direta com os dados reais. A métrica é representada pela equação:

$$RMSE = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (yi - \hat{y}i)^2}$$

O Coeficiente de Determinação (R^2) [27] é uma métrica que avalia o quanto a variabilidade dos dados reais é explicada pelo modelo, ela indica o grau de ajuste do modelo em relação aos dados observados. Essa métrica varia entre 0 e 1, onde valores mais próximos de 1 representam modelos com melhor capacidade de explicar os resultados reais. Essa métrica é representada pela seguinte equação:

$$R^2 = 1 - (\Sigma (yi - \hat{y}i)^2 / \Sigma (yi - \bar{y})^2),$$

onde $\bar{y}$ é a média dos valores reais.

4.11 Ameaças a Validade

Algumas ameaças cerceiam esse estudo, dentre elas:

- **Validade interna - Viés de Seleção:** A coleta dos relatórios de bugs dentro dos requisitos descritos no escopo deste estudo pode ter introduzido tendência dos dados a tipos predeterminados de relatórios de bugs com problemas atuais devido aos avanços decorrentes das novas tecnologias. Com isso, expandimos os filtros de busca para considerarem períodos maiores para diminuir assim os riscos. Além disso, a classificação manual dos comentários dos relatórios de bugs pode ter adicionado viés pelo autor ao julgar única e solitariamente os comentários aos tipos definidos na etapa de classes e classificação de comentários de relatórios de bugs. Para mitigar esse possível viés, foi utilizada uma interface auxiliar durante a rotulação manual dos comentários, que apresentava ao autor o comentário em conjunto com seu contexto completo, incluindo comentários anteriores e posteriores no mesmo relatório, além de informações associadas como presença de anexos e referências. Essa visualização ampliada permitiu uma tomada de decisão mais consistente e informada no momento da atribuição das classes.
- **Validade interna - Critérios de Filtragem:** A inserção de critérios para seleção mais rigorosa dos relatórios de bugs a serem colhidos pode acabar incluindo vieses relacionados aos tipos de relatórios que foram retornados, realizando implicitamente, uma padronização dos dados utilizados para o estudo.
- **Validade externa - Generalização dos Resultados:** A seleção de relatórios de bugs somente de uma fonte de dados do Bugzilla pode introduzir o risco de generalização para os resultados que forem obtidos a partir desses dados. O risco de generalização diz respeito principalmente ao inferir que os resultados obtidos para os dados de um repositório podem refletir a realidade em outros repositórios do Bugzilla.

5 Taxonomia

Este capítulo apresenta o processo de definição da taxonomia de classes de comentários e a metodologia adotada para a classificação manual dos comentários extraídos dos relatórios de bugs.

5.1 Definição das Classes de Comentários e Anexos

As classes utilizadas neste estudo foram definidas com o propósito de organizar e classificar os comentários presentes nos relatórios de bugs de acordo com o seu conteúdo textual e função no processo de resolução dos relatórios de bugs. Essa classificação possibilita diferenciar comentários que possuem contribuições técnicas e informativas daqueles que exercem funções mais comunicativas ou complementares.

Neste trabalho, o termo “classe” é utilizado em dois sentidos complementares. No contexto da Figura 10, o termo refere-se às classes do modelo conceitual representado em UML/MOF, utilizadas para estruturar formalmente a taxonomia proposta. No contexto do processo de classificação dos comentários, o termo “classe” refere-se às categorias conceituais empregadas para rotular e agrupar comentários de acordo com seu conteúdo textual e função no relatório de bug. Embora representadas como classes no modelo conceitual, essas categorias devem ser interpretadas como classes classificatórias, e não como classes de implementação de software.

A definição das classes de comentários foi realizada de forma manual, pelo autor, a partir de uma análise exploratória de uma amostra inicial de comentários extraídos do próprio *dataset* de comentários utilizado neste estudo. Esse processo envolveu a leitura sistemática dos comentários e a identificação de padrões recorrentes de conteúdo e intenção comunicativa, resultando na proposição de classes abrangentes, capazes de acomodar diferentes variações textuais sem gerar distinções excessivamente finas entre categorias.

A criação das classes foi motivada pela necessidade de compreender como diferentes tipos de interações podem ocorrer na seção de comentários durante o ciclo de

vida de um relatório de bug e de que forma essas interações influenciam na eficiência da resolução. Cada comentário foi analisado e classificado de acordo com a classe correspondente, permitindo que essas informações fossem posteriormente utilizadas como variáveis nos modelos de *machine learning*. Dessa forma, tornou-se possível investigar relações entre os tipos de comentários observados e os tempos de resolução dos relatórios de bugs.

Como resultado dessa etapa, foram definidas dez classes principais de comentários que representam as formas mais recorrentes de contribuição textual observadas em relatórios de bugs. Os tipos de anexos foram incluídos nesse sistema de classificação por estarem diretamente associados aos comentários nos quais são inseridos, funcionando como elementos complementares ao conteúdo textual e, frequentemente, fornecendo evidências técnicas adicionais relevantes para a compreensão e resolução dos problemas relatados. Para isso, foram definidos cinco tipos de anexos que podem estar associados a esses comentários, abrangendo desde arquivos de código e capturas de tela, até links externos e referências a outros relatórios. Cada comentário e anexo foi identificado de forma individual, possibilitando análises combinadas sobre a influência do conteúdo textual e dos recursos anexados no andamento e desfecho dos relatórios de bugs.

A validação das classes propostas foi realizada de forma qualitativa ao longo da etapa de classificação manual. À medida que novas classes eram definidas, os comentários previamente classificados eram revisados para verificar a consistência e a adequação da categorização, garantindo que cada comentário estivesse associado à classe que melhor representasse seu conteúdo e função. Esse processo iterativo contribuiu para reduzir ambiguidades entre classes e reforçar a coerência interna da taxonomia adotada. Além disso, a aplicação das classes em um conjunto amplo e diverso de comentários permitiu verificar sua capacidade de generalização, assegurando que o conjunto final de classes fosse suficientemente expressivo para representar os principais tipos de interações observadas em relatórios de bugs.

O próximo subcapítulo 5.1.1 de classes descreve melhor as classes de comentários e anexos.

Classes

Este subcapítulo descreve os conceitos e classes utilizados para a classificação dos tipos de comentários presentes em relatórios de bugs. A taxonomia proposta está organizada a partir de um modelo conceitual que relaciona relatórios de bugs, comentários e os tipos de conteúdo e anexos associados a esses comentários, conforme ilustrado na Figura 10.

No topo da estrutura conceitual encontra-se a classe Bug Report, que representa um relatório de bug em um sistema de rastreamento. Cada relatório de bug pode possuir zero ou mais comentários, representados pela classe *Comment*, refletindo as interações que ocorrem ao longo do ciclo de vida do relatório. Cada comentário está associado a um único relatório de bug, caracterizando uma relação de cardinalidade *1–0..* entre *Bug Report* e *Comment*.

A classe *Comment* representa a unidade central de análise deste estudo, uma vez que os comentários constituem o principal artefato textual investigado. Cada comentário pode conter informações textuais, anexos ou ambos, sendo classificado de acordo com dois eixos conceituais principais: o tipo de conteúdo textual e o tipo de anexo associado. Esses dois eixos são representados pelas classes abstratas *Content Type* e *Attachment*, que servem como generalizações para as subclasses responsáveis por caracterizar os diferentes tipos de comentários observados nos relatórios de bugs.

Classe *Content Type*

A classe *Content Type* determina o tipo de conteúdo textual predominante em um comentário. Essa classificação é realizada com base na análise do texto do comentário e no papel que ele desempenha no processo de análise e resolução do relatório de bug. Cada comentário é associado a exatamente um tipo de conteúdo, caracterizando uma classificação exclusiva.

Conforme ilustrado na Figura 10, a classe *Comment* está associada a uma instância da classe *Content Type*, indicando que todo comentário possui um único tipo

de conteúdo atribuído. A classe *Content Type* é abstrata e define uma hierarquia de subclasses que representam as formas mais recorrentes de contribuição textual identificadas nos relatórios de bugs analisados.

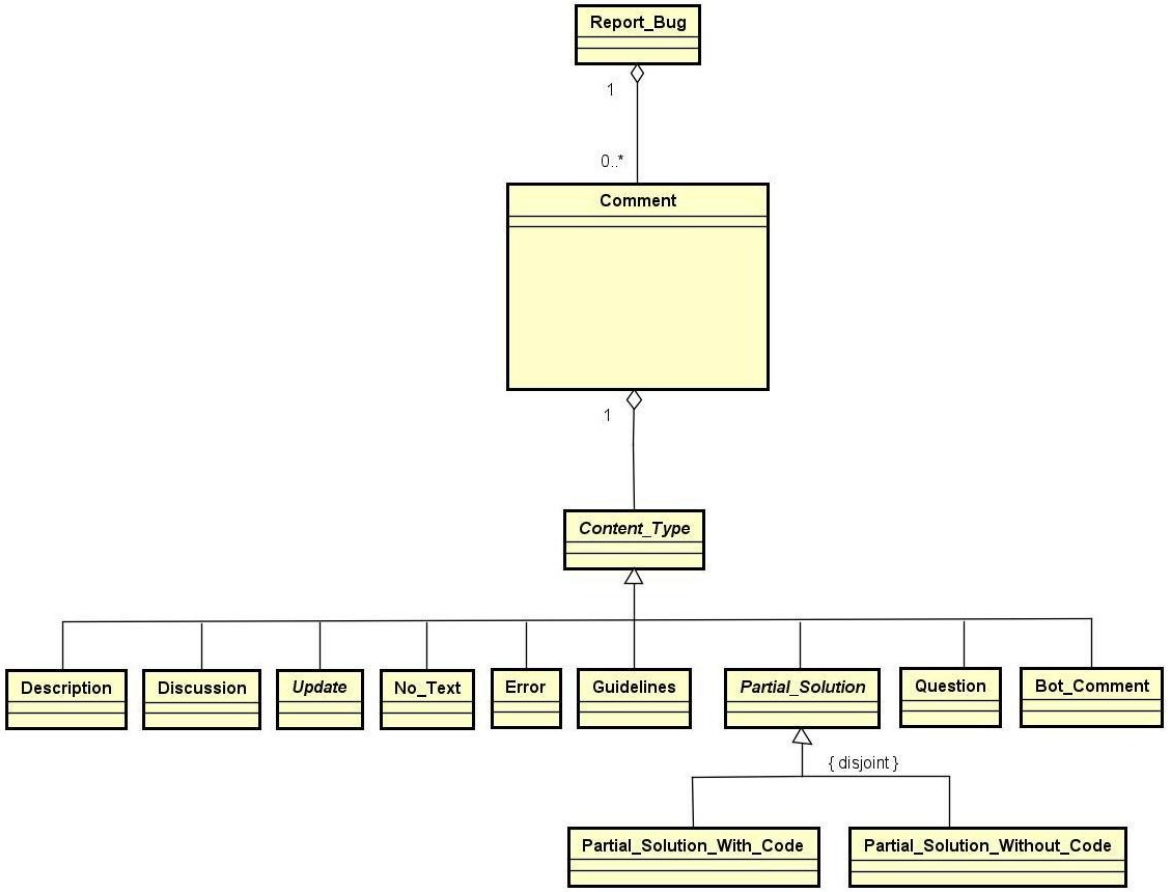


Figura 10: Representação visual em MOF da classe Content Type da taxonomia.

As subclasses que compõem a classe *Content Type* no modelo conceitual correspondem às classes classificatórias utilizadas neste estudo para rotular os comentários. Essas classes são: *Description*, *Discussion*, *Error*, *Guideline*, *Partial Solution*, *Update*, *No Text*, *Question* e *Bot Comment*. A classe *Partial Solution* é especializada em duas subclasses mutuamente exclusivas: *Partial Solution With Code* e *Partial Solution Without Code*, permitindo distinguir comentários que apresentam trechos de código daqueles que descrevem soluções apenas de forma textual.

Essa estrutura hierárquica possibilita uma tipagem consistente dos comentários, fornecendo uma base conceitual para a aplicação dos modelos de machine learning utilizados neste estudo.

Description

A classe *Description* abrange comentários que correspondem à descrição inicial de um relatório de bug. Esses comentários são, em geral, o primeiro registro textual do relatório e têm como objetivo apresentar de forma estruturada o problema observado, fornecendo contexto suficiente para que outros participantes compreendam a natureza do defeito reportado. Os comentários classificados como *Description* normalmente incluem informações como a descrição do comportamento incorreto do sistema, o ambiente em que o problema ocorreu, mensagens de erro, rastreamentos de pilha (*stack traces*), exemplos de execução e, quando possível, indícios sobre a causa do problema. Diferentemente de comentários subsequentes, essa classe não representa uma interação ou diálogo, mas sim a formalização inicial do defeito que motiva a abertura do relatório.

Na classificação manual realizada neste estudo, os comentários associados a essa classe classificatória recebem a etiqueta de identificação ‘DE’, por combinarem aspectos descritivos e, frequentemente, a explicitação de falhas ou erros observados no sistema. O comentário ilustrado na Figura 11 foi classificado como *Description* por apresentar a descrição detalhada do problema enfrentado, contextualizando o erro ocorrido, relatando o comportamento inesperado do sistema e fornecendo informações técnicas relevantes, como mensagens de exceção e rastreamento de pilha, sem estabelecer ainda uma troca de informações com outros participantes. Esse tipo de comentário é fundamental para o processo de resolução de relatórios de bugs, pois serve como base para a análise inicial do problema, orientando investigações posteriores, solicitações de esclarecimento e propostas de solução ao longo do ciclo de vida do relatório.



James Teh [jamie] (away returning 12 Jan) Reporter
Description - 5 years ago

The third party, open-source NVDA screen reader injects dlls into Firefox. To help with debugging of related crashes, symbols are uploaded to Mozilla as part of the automated build process. Recently, this started failing, It turns out that this was due to an expired API token. I've extended this now, so this should be fixed. However, the cause of the problem was not at all obvious. Instead of reporting an HTTP error code, the connection was simply dropped. This resulted in the following traceback from the Python requests library:

```
Traceback (most recent call last):
   File "appveyor\MozillaSyms.py", line 103, in <module>
     upload()
   File "appveyor\MozillaSyms.py", line 87, in upload
     allow_redirects=False
   File "C:\Python37\lib\siite-packages\requests\api.py", line 119, in post
     return requests('post', url, data-data, json=json, **kwargs)
   File "C:\Python37\lib\site-packages\requests\api.py", line 61, in request
     return requests.request(method=method, url=url, **kwargs)
   File "C:\Python37\lib\site-packages\requests\sessions.py", liine 543,
in send
     r = adapter.send(prep, **send_kwargs)
   File "C:\Python37\lib\site-packages\rrequests\sessions.py", line 643,
in send
     r = adapter.send(request, **kwargs)
   File "C:\Python37\lib\site-packages\rrequests\adapters.py", line 498,
in send
     raise ConnectionError(err, request=request)
requests.exceptions.ConnectionError: ('Connection aborted.',
RemoteDisconnected('Remote end closed connection without respoonse'))
```

When the token has expired, it would be really helpful if an HTTP error code could be reported along with an explanatory message, rather than dropping the connection without a response.

Thanks!

Figura 11: *Comentário do tipo Description retirado do relatório de bug* ***ID: 1655944*** *[13].*

Discussion

A classe *Discussion* abrange comentários cujo principal objetivo é a troca de informações, esclarecimentos ou coordenação de ações relacionadas ao relatório de bug. Esses comentários caracterizam interações conversacionais que podem ocorrer entre desenvolvedores, usuários, ou outros participantes do processo de manutenção, não se restringindo exclusivamente a desenvolvedores. Em geral, comentários desta classe não apresentam soluções completas nem instruções técnicas detalhadas, mas contribuem

para o entendimento do problema, para o acompanhamento do progresso ou para a solicitação de informações adicionais.

Na classificação manual realizada neste estudo, os comentários classificados com esse tipo recebem uma etiqueta de identificação ‘D’. O comentário ilustrado na Figura 12 foi classificado como *Discussion* por apresentar uma interação típica de diálogo, na qual o autor reconhece o problema relatado, comenta o comportamento observado e solicita informações adicionais, sem propor diretamente uma solução técnica ou correção definitiva. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 1655944 [13], e ilustra uma situação recorrente em que o comentário tem caráter predominantemente comunicativo e investigativo, justificando sua classificação na classe *Discussion*.

É importante destacar que nem todo comentário subsequente à descrição inicial foi automaticamente classificado como *Discussion*. A classificação considerou o propósito predominante do conteúdo textual. Comentários que apresentavam soluções parciais, instruções técnicas, atualizações formais ou questionamentos objetivos foram classificados em suas respectivas categorias, mesmo que configurassem uma resposta dentro da thread. Assim, a classe *Discussion* foi atribuída apenas quando o comentário tinha caráter predominantemente comunicativo, investigativo ou de coordenação.

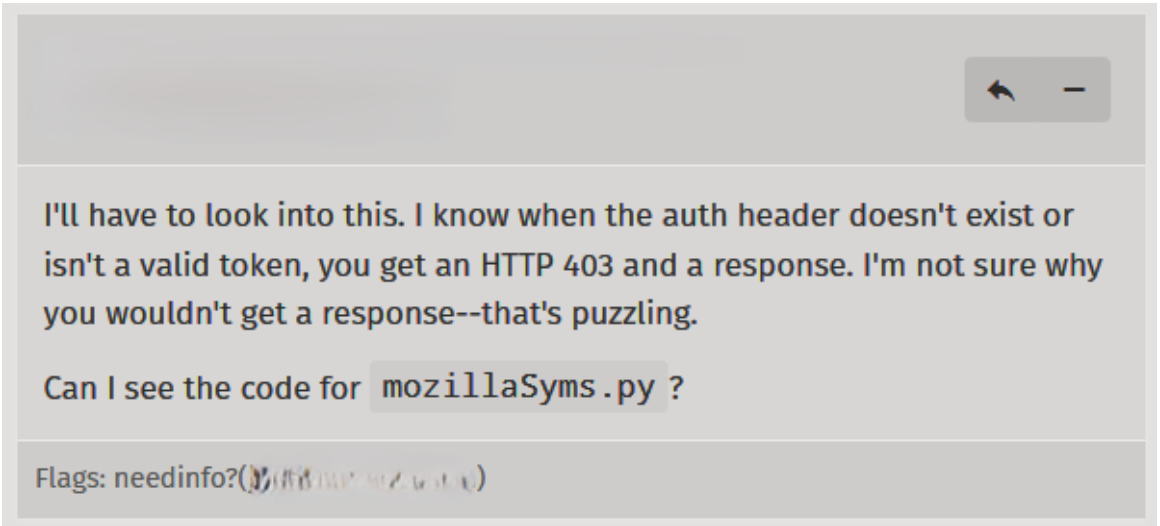


Figura 12: Comentário do tipo Discussion retirado do relatório de bug ***ID: 1655944*** *[13].*

Error

A classe *Error* abrange os comentários que contêm informações sobre erros que apareceram durante o processo de replicação do bug, e ou, durante as tentativas de correção do problema. Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação ‘E’. O comentário ilustrado na Figura 13 foi classificado como *Error* por relatar explicitamente a ocorrência de um erro durante o uso de um token expirado, apresentando tanto o código de retorno HTTP quanto a mensagem de erro associada, além de contextualizar o problema no ambiente de execução da aplicação. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 1655944 [13], e ilustra um caso típico em que o comentário tem caráter predominantemente informativo e técnico, descrevendo uma falha observada sem, contudo, propor uma solução definitiva, o que justifica sua classificação na classe *Error*.

No contexto deste estudo, o termo *Error* refere-se à manifestação observável do problema relatado, como mensagens de erro, códigos de retorno HTTP, exceções ou comportamentos incorretos percebidos durante a execução do sistema. Não se trata necessariamente do defeito interno no código-fonte que originou o problema, mas sim da evidência técnica reportada nos comentários. A adoção do termo *Error* está alinhada à terminologia comumente utilizada nos próprios relatórios do Bugzilla, onde desenvolvedores e usuários descrevem a ocorrência de erros visíveis como parte do processo de investigação.



Figura 13: Comentário do tipo Error retirado do relatório de bug ***ID: 1655944*** *[13].*

Guideline

A classe *Guideline* abrange os comentários que contenham informações adicionais para resolução do problema, e ou, informações relevantes que auxiliem na resolução do problema informado no relatório. Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação 'G'. O comentário ilustrado na Figura 14 foi classificado como *Guideline* por apresentar uma referência direta a um trecho de código utilizado anteriormente, disponibilizando um link para um repositório externo com uma versão alternativa da implementação. Esse tipo de comentário tem como principal objetivo orientar outros participantes do relatório, oferecendo subsídios técnicos que podem facilitar a análise ou a correção do problema. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 1655944 [13], e ilustra um caso recorrente em que o comentário atua como um guia técnico, fornecendo informações adicionais relevantes sem, contudo, configurar uma solução completa ou definitiva, o que justifica sua classificação na classe *Guideline*.

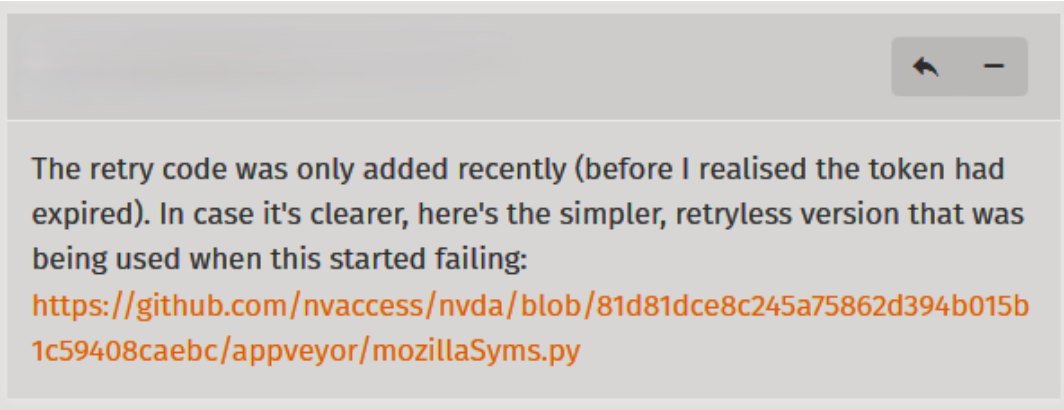


Figura 14: Comentário do tipo Guideline retirado do relatório de bug ***ID: 1655944*** *[13].*

Partial Solution

A classe *Partial Solution* se configura como uma superclasse abstrata que agrupa comentários que apresentam contribuições técnicas com potencial para avançar a resolução do problema descrito no relatório de bug, mas que não caracterizam, isoladamente, uma solução completa ou definitiva. Esses comentários geralmente descrevem ideias, abordagens, trechos de código ou sugestões técnicas que auxiliam no processo de correção, porém dependem de validação adicional, complementação ou integração com outras etapas do desenvolvimento.

A adoção do termo *Partial Solution* decorre da própria natureza colaborativa dos relatórios de bugs, nos quais a resolução completa do problema raramente é apresentada em um único comentário. Em vez disso, a solução emerge progressivamente a partir de múltiplas interações, testes e refinamentos. Assim, mesmo comentários tecnicamente relevantes são classificados como soluções parciais quando não encerram o processo de correção do bug.

A classe *Partial Solution* é especializada em duas subclasses mutuamente exclusivas: *Partial Solution With Code* e *Partial Solution Without Code*, permitindo distinguir comentários que apresentam trechos de código daqueles que descrevem soluções de forma exclusivamente textual.

Partial Solution With Code

A classe de *Partial Solution With Code* envolve comentários que propõem soluções incompletas que provoquem avanços na resolução do problema, e que contenham código fonte junto ao texto do comentário. Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação 'PC'. O comentário ilustrado na Figura 15 foi classificado como *Partial Solution With Code* por apresentar uma sugestão de modificação no código responsável pela limpeza do cache, incluindo um trecho de código e uma observação técnica sobre a forma adequada de realizar a operação. Essa contribuição caracteriza uma solução parcial, uma vez que propõe uma correção específica, porém ainda depende de validação, testes adicionais e possível integração com outras alterações antes de ser considerada uma solução completa.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955957 [18], e ilustra um padrão recorrente observado nos dados analisados, no qual comentários contendo código atuam como etapas intermediárias no processo de resolução colaborativa de bugs. Esse tipo de comentário desempenha um papel fundamental na evolução da solução, justificando sua classificação como *Partial Solution With Code*.

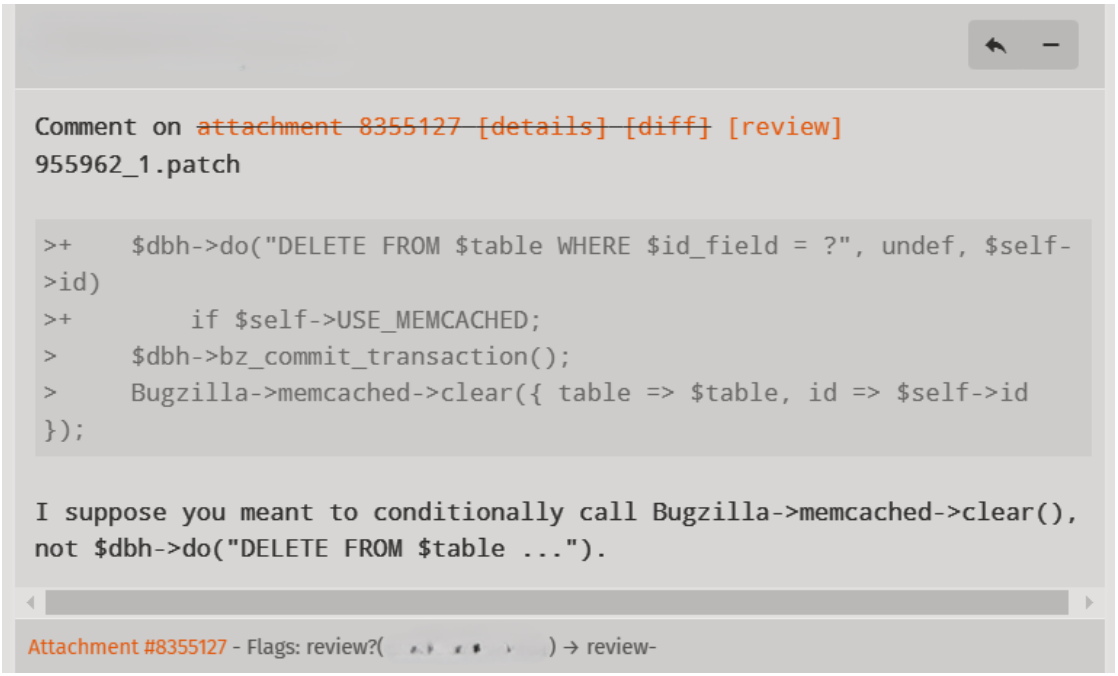


Figura 15: Comentário do tipo Partial Solution With Code retirado do relatório de bug ***ID: 955957*** *[18].*

Partial Solution Without Code

A classe *Partial Solution Without Code* abrange comentários que propõem soluções incompletas que alavanquem a resolução do problema mensurado no relatório, mas que não contenham código fonte escrito no seu texto. Na classificação manual realizada nestes estudos, os comentários classificados com esse tipo recebem uma etiqueta de identificação 'POC'. A etiqueta 'POC' foi adotada por representar de forma sucinta o conceito de *Proof of Concept* [67], amplamente utilizado na engenharia de software para designar propostas técnicas que descrevem abordagens ou direções de solução sem necessariamente apresentar uma implementação completa.

O comentário ilustrado na Figura 16 foi classificado como *Partial Solution Without Code* por descrever, de forma detalhada, quais arquivos e componentes do sistema deveriam ser modificados, além de indicar referências externas e exemplos de comportamento esperado, sem apresentar diretamente a implementação em código. Essa característica evidencia uma contribuição técnica relevante, porém ainda incompleta, justificando sua classificação como uma solução parcial sem código.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 956044 [13], e ilustra um padrão recorrente observado nos dados analisados, no qual o autor do comentário fornece orientações técnicas claras sobre como proceder com a correção, delegando a implementação efetiva a etapas posteriores do processo de desenvolvimento.

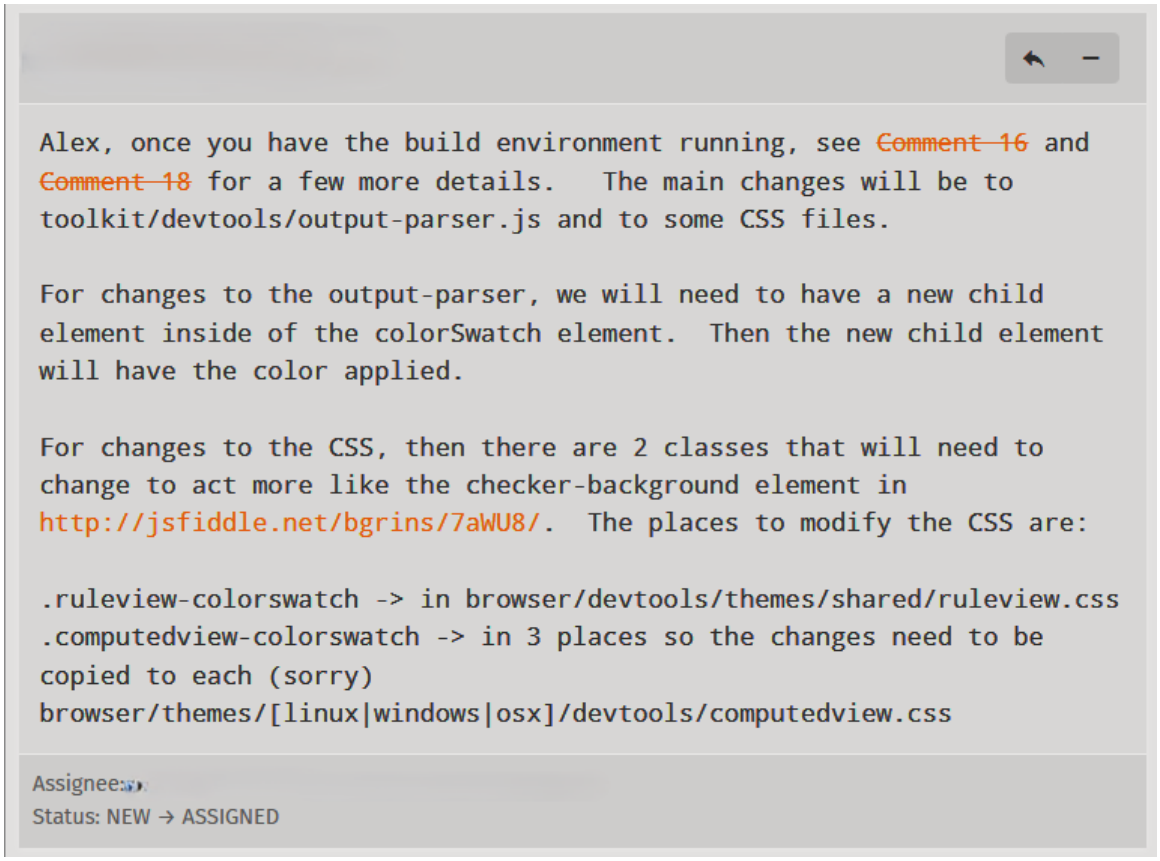


Figura 16: Comentário do tipo Partial Solution Without Code retirado do relatório de bug ***ID: 956044*** *[19].*

Update

A classe *Update* abrange comentários cujo principal objetivo é registrar atualizações relevantes no estado do relatório de bug, incluindo mudanças explícitas de status, resolução ou campos associados ao relatório, bem como comunicações que informam avanços significativos no andamento do problema. Essas atualizações podem ocorrer de forma automática, refletidas diretamente nos campos do relatório, ou de forma descritiva, quando o comentário comunica alterações importantes sem necessariamente acionar uma mudança imediata de status.

Na classificação manual realizada neste estudo, os comentários pertencentes a essa classe recebem a etiqueta ‘U’. O comentário ilustrado na Figura 17 foi classificado como *Update* por informar a implantação da correção em ambiente de produção e pela consequente mudança do estado do relatório para *FIXED*, caracterizando uma atualização formal e conclusiva do problema.

Além de mudanças de status, também foram classificados como *Update* comentários que comunicam alterações relevantes no entendimento do problema, no seu progresso ou em campos descritivos do relatório, desde que essas informações representam uma atualização objetiva do estado do bug, e não apenas uma discussão ou sugestão técnica. Dessa forma, a classe *Update* captura comentários que atuam como marcos informativos ao longo do ciclo de vida do relatório de bug.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 1655944 [13], e ilustra um padrão recorrente no qual comentários de atualização sinalizam a transição do relatório para uma nova etapa do processo de resolução.

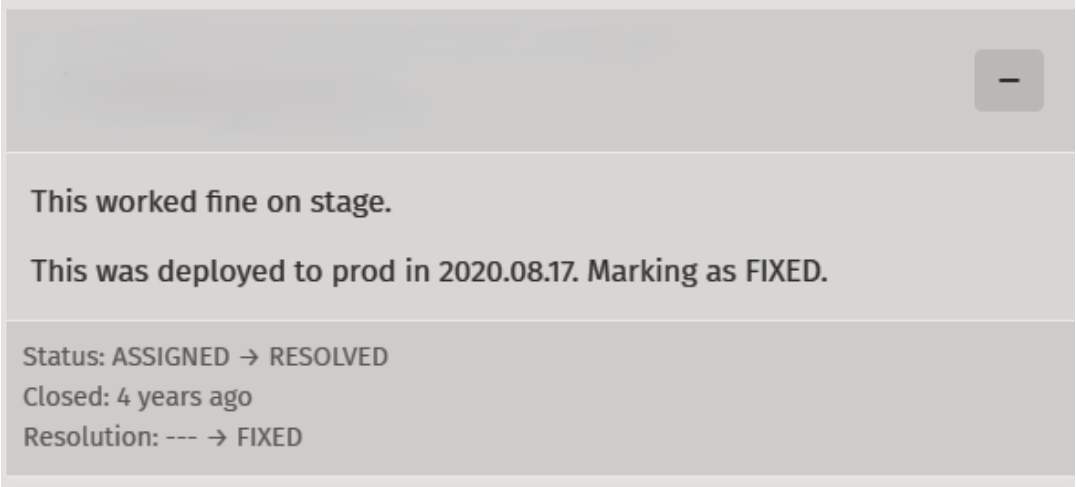


Figura 17: Comentário do tipo Update retirado do relatório de bug ***ID: 1655944*** *[13].*

No Text

A classe *No Text* identifica os comentários que não apresentam conteúdo textual significativo, ou seja, não contêm informações descritivas, explicativas ou argumentativas escritas em linguagem natural. Esses comentários podem consistir exclusivamente em referências externas, como links, identificadores de revisões ou outros tipos de anexos, sem qualquer contextualização textual adicional.Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação 'N'. O comentário ilustrado na Figura 18 foi classificado como *No Text* por não conter texto explicativo, sendo composto apenas por uma sequência de links

associados ao relatório de bug. Ainda assim, esse tipo de comentário pode ser considerado válido para o objeto de estudo quando os anexos fornecidos contribuem para a compreensão ou resolução do problema, como no caso de referências a commits ou revisões de código.

Por outro lado, comentários que não apresentam conteúdo textual relevante nem anexos informativos foram considerados não contributivos e, portanto, passíveis de exclusão das análises subsequentes. Dessa forma, a classe *No Text* permite distinguir entre comentários semanticamente vazios e comentários que, apesar da ausência de texto, desempenham um papel informativo por meio de anexos associados.

O exemplo apresentado foi extraído do relatório de bug de ID 955888 [16] do sistema Bugzilla, e ilustra um padrão recorrente observado nos dados analisados, no qual comentários sem texto estruturado ainda podem conter informações úteis para o processo de resolução do relatório.

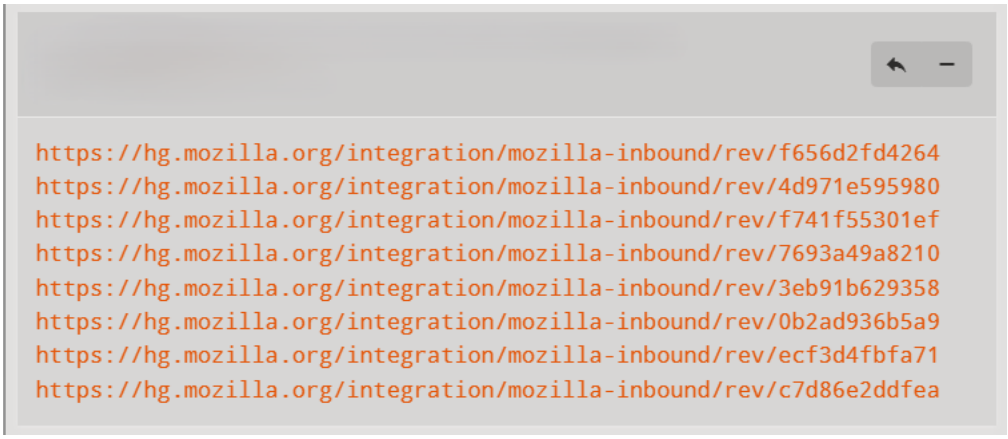


Figura 18: Comentário do tipo No Text retirado do relatório de bug ***ID: 955888*** *[16].*

Question

A classe *Question* identifica comentários que sejam explicitamente perguntas direcionadas ao autor do relatório de bug ou aos demais participantes envolvidos no processo de análise e resolução. Esses comentários têm como objetivo solicitar esclarecimentos, confirmar hipóteses ou obter informações adicionais necessárias para

o entendimento ou progresso do relatório. Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação ‘Q’.

O comentário ilustrado na Figura 19 foi classificado como *Question* por consistir exclusivamente em uma pergunta direta sobre o comportamento de um trecho específico do código, caracterizando de forma clara a intenção de solicitar esclarecimento técnico. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 956066 [20], e ilustra um padrão recorrente observado nos dados analisados, no qual comentários interrogativos atuam como gatilhos para a obtenção de informações adicionais, frequentemente antecedendo discussões, atualizações ou propostas de solução ao longo do ciclo de vida do relatório.

É importante destacar que a simples presença de uma pergunta em um comentário não foi critério suficiente para classificá-lo como Question. Diversas classes podem conter questionamentos embutidos como parte de uma explicação, proposta de solução ou discussão técnica. A classificação considerou o propósito predominante do comentário no contexto da thread de comentários. Assim, a etiqueta ‘Q’ foi atribuída apenas quando o objetivo principal do comentário era solicitar esclarecimento ou informação adicional, sem apresentar simultaneamente uma solução estruturada, instrução técnica detalhada ou atualização formal do relatório.

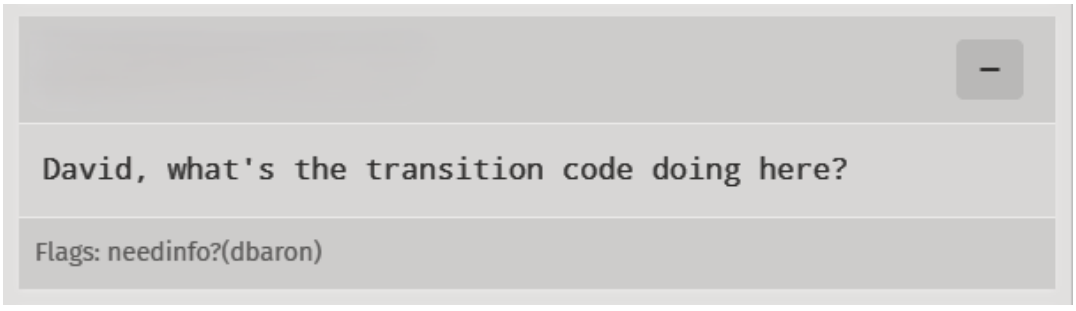


Figura 19: Comentário do tipo Question retirado do relatório de bug ***ID: 956066*** *[20].*

Bot Comment

A classe *Bot Comment* identifica comentários gerados automaticamente por bots ou robôs de integração contínua que atuam no ambiente do Bugzilla ou em ferramentas externas integradas ao seu fluxo. Neste estudo, os termos *bots* e *robôs* são utilizados de

forma equivalente, ambos se referem a agentes automatizados responsáveis por executar tarefas previamente definidas, como a execução de testes, validações de integração contínua ou o registro automático de resultados no sistema de rastreamento de bugs. Não é feita distinção conceitual entre *bots* e *robôs*, uma vez que, no contexto do Bugzilla, e das ferramentas a ele integradas, ambos desempenham a mesma função operacional: a geração automática de comentários sem intervenção humana direta.

Esses comentários não são produzidos por usuários humanos e têm como finalidade registrar resultados de testes automatizados, falhas de compilação, execuções de pipelines ou outras informações geradas durante o processo automático de validação do código. Na classificação manual, os comentários classificados com esse tipo recebem uma etiqueta de identificação 'B'. O comentário ilustrado na Figura 20 foi classificado como *Bot Comment* por ser explicitamente identificado como proveniente de um robô (Legacy TBPL/Treeherder Robot), apresentando informações padronizadas sobre ambiente de execução, versão do código, status do teste e resultado da execução, sem qualquer intervenção textual humana.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955934 [22], e ilustra um padrão recorrente observado nos dados analisados, no qual comentários automatizados atuam como mecanismos de monitoramento e validação contínua, fornecendo informações técnicas relevantes para o acompanhamento do relatório, mas sem caráter discursivo ou interpretativo.



Legacy TBPL/Treeherder Robot
Comment 1 · 11 years ago

philor
https://tbpl.mozilla.org/php/getParsedLog.php?id=33349025&tree=Mozilla-Inbound
Ubuntu VM 12.04 x64 mozilla-inbound opt test mochitest-browser-chrome on 2014-01-21 11:39:37
revision: 3b8da3495a3
slave: TST-UNEXPECTD-FAIL
chrome://mochitest/content/browser/browser/devtools/styleinspector/test/browser_bug946331_close_tooltuip_on_new_selection.js
Test timed out

Figura 20: Comentário do tipo Bot Comment retirado do relatório de bug ***ID: 955934*** *[22].*

Classe *Attachment*

A classe *Attachment* identifica os arquivos, hiperlinks e demais recursos anexados aos comentários em relatórios de bugs, os quais complementam ou substituem o conteúdo textual apresentado. Diferentemente da classificação baseada no conteúdo textual, representada pela classe *Content Type*, a classificação de anexos é secundária e opcional: todo comentário possui obrigatoriamente um tipo de conteúdo, mas pode possuir zero ou mais anexos associados.

Conforme ilustrado na Figura 21, a classe *Comment* mantém um relacionamento de cardinalidade *0..** com a classe *Attachment*, indicando que um comentário pode não possuir anexos ou conter múltiplos anexos de naturezas distintas. A classe *Attachment* é definida como uma superclasse abstrata que organiza esses recursos em subclasses específicas, permitindo caracterizar de forma estruturada os tipos de informações adicionais fornecidas nos comentários.

A inclusão dessa hierarquia no modelo conceitual é motivada pelo papel relevante que os anexos desempenham no processo de resolução de relatórios de bugs. Arquivos, links, documentações, stack traces e referências a relatórios semelhantes frequentemente fornecem evidências técnicas, contexto adicional ou caminhos

alternativos para a compreensão e correção do problema, mesmo quando o comentário textual é limitado ou inexistente.

É importante destacar que a hierarquia Attachment modela exclusivamente os anexos estruturais formalmente registrados na plataforma Bugzilla, identificados por meio do campo *attachment_id*. Elementos como links, stack traces ou referências a outros relatórios que apareçam apenas no corpo textual do comentário não são representados como instâncias da classe Attachment, mas sim capturados indiretamente pela análise textual realizada na etapa de processamento de linguagem natural. Essa decisão metodológica foi adotada para distinguir entre recursos explicitamente anexados ao relatório (artefatos estruturados) e informações incorporadas ao texto livre do comentário. Embora tais elementos textuais também possam influenciar o tempo de resolução de um relatório, sua modelagem foi tratada no âmbito das features textuais, e não como especializações formais da classe Attachment.

No contexto deste trabalho, os tipos de *Attachment* não são utilizados como classes-alvo nos modelos de machine learning para classificação de comentários. Em vez disso, as informações relacionadas à presença e ao tipo de anexos são exploradas como features auxiliares no processo de classificação e análise dos relatórios de bugs. Dessa forma, a hierarquia *Attachment* contribui para enriquecer a representação dos relatórios, permitindo investigar como diferentes tipos de anexos se relacionam com o tempo de resolução e com a evolução do status dos bugs. A especialização adicional de recursos textuais embutidos no comentário, como a distinção automática entre links externos, referências bibliográficas ou trechos de código, constitui uma possibilidade de aprofundamento futuro, que exigiria técnicas específicas de extração semântica e análise estrutural do texto.

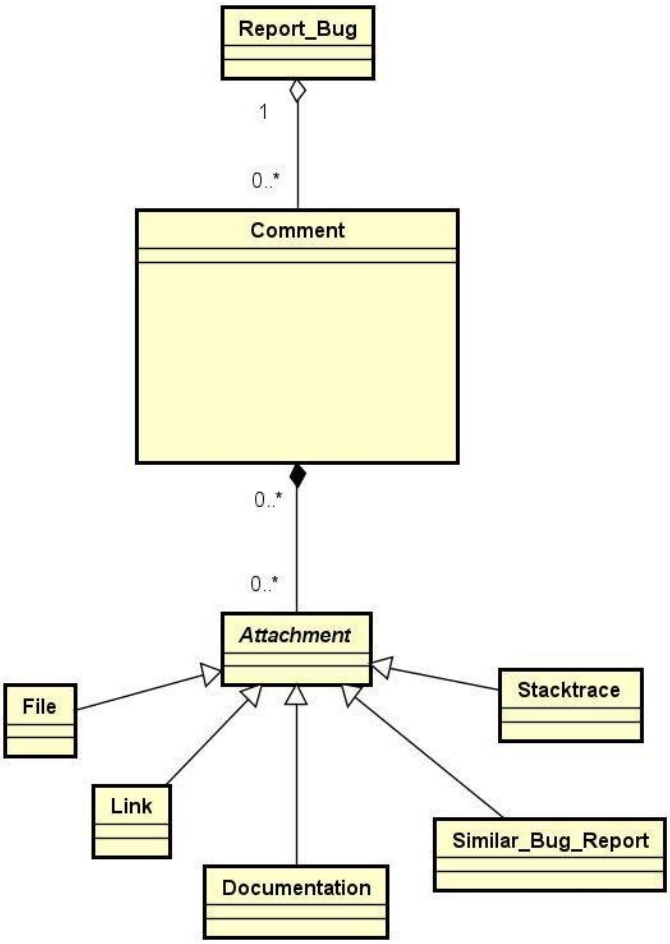


Figura 21: Representação visual em MOF da classe Attachment da taxonomia.

Assim, a presença de anexos estruturais é tratada como uma variável potencialmente associada à dinâmica de resolução dos relatórios, sendo considerada no conjunto de atributos utilizados nos experimentos preditivos descritos na Seção 4.8. As subclasses que compõem a classe *Attachment* são: *File*, *Link*, *Documentation*, *Similar Bug Report* e *Stacktrace*. Cada uma dessas subclasses representa um tipo específico de recurso anexado a comentários e será detalhada nas subseções a seguir.

File

A classe *File* abrange os anexos que correspondem a arquivos explicitamente associados a comentários em relatórios de bugs, tais como patches, imagens, documentos ou outros artefatos binários disponibilizados por meio do sistema de rastreamento. Esses anexos são armazenados como arquivos no próprio repositório do

Bugzilla e ficam vinculados ao comentário que os referência. Na classificação manual os comentários que possuem anexos deste tipo recebem uma etiqueta de 'Attachment' com 'AF'. O comentário ilustrado na Figura 22 foi classificado como *File* por conter um arquivo anexado ao relatório, identificado explicitamente como um *attached file*, representando um artefato técnico associado ao processo de correção do bug.

É importante ressaltar que, no contexto deste trabalho, a classe *File* refere-se à existência e ao tipo do anexo vinculado ao comentário, independentemente do conteúdo específico do arquivo ou de sua análise interna. Os arquivos não precisam estar disponíveis localmente nem serem processados diretamente pelos modelos de machine learning; sua presença é considerada como uma característica informativa, utilizada como feature auxiliar na análise dos relatórios de bugs.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 1655944 [13], e ilustra um padrão recorrente em que arquivos anexados fornecem subsídios técnicos relevantes, como correções propostas ou evidências do problema, contribuindo para o entendimento e a evolução do relatório.

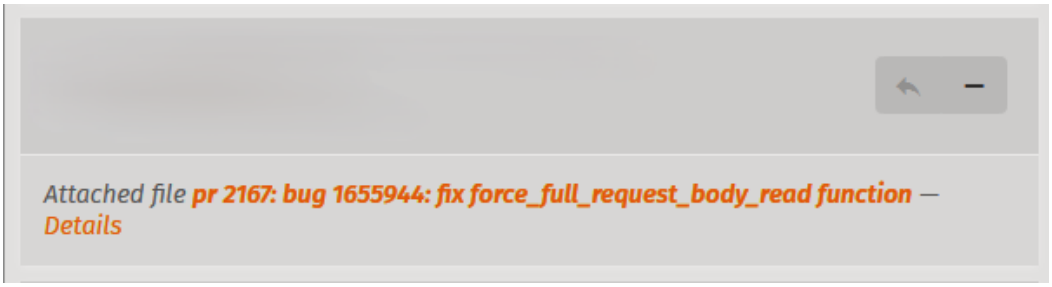


Figura 22: Comentário com Attachment do tipo File retirado do relatório de bug ***ID: 1655944*** *[13].*

Link

A classe *Link* abrange anexos representados por hiperlinks presentes no corpo de comentários em relatórios de bugs. Esses links podem direcionar para diferentes tipos de recursos externos, como repositórios de código, revisões (*commits*), sistemas de integração contínua, documentações técnicas ou outros artefatos relevantes ao processo de análise e resolução do problema. Na classificação manual, os comentários que

possuem anexos deste tipo recebem uma etiqueta de ‘Attachment’ com ‘AL’. A Figura 23 ilustra um comentário classificado com *Attachment* do tipo *Link*, no qual não há conteúdo textual explicativo, mas há a presença de um hiperlink que referência diretamente uma revisão de código externa ao sistema de rastreamento.

Nesse contexto, o comentário foi classificado como *No Text* quanto ao seu conteúdo textual, mas considerado válido para o estudo devido à presença de um *Attachment* do tipo *Link*. Essa distinção reforça a separação conceitual entre a classificação baseada no conteúdo textual (*Content Type*) e a classificação baseada em anexos (*Attachment*), permitindo capturar contribuições relevantes mesmo na ausência de texto em linguagem natural.

O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955888 [16], e ilustra um padrão recorrente observado nos dados analisados, no qual links funcionam como mecanismos de referência externa, fornecendo informações técnicas úteis para o acompanhamento e a resolução do relatório.

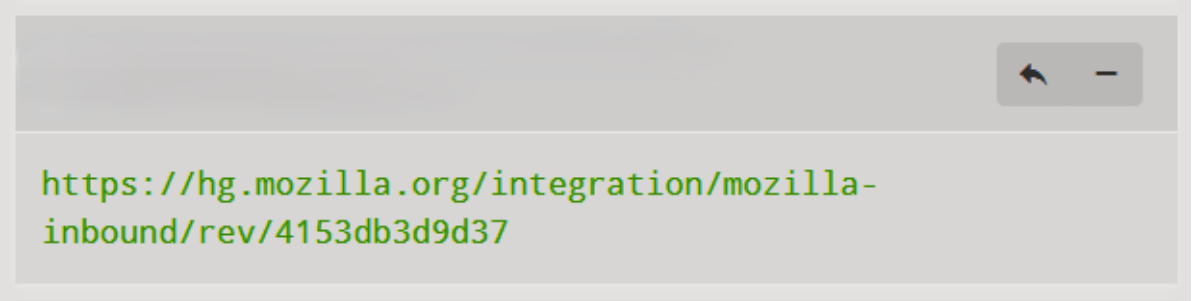


Figura 23: Comentário com Attachment do tipo Link retirado de relatório de bug ***ID: 955888*** *[16].*

Documentation

A classe *Documentation* abrange anexos representados por links que direcionam para páginas de documentação técnica, sejam elas relacionadas ao código-fonte utilizado na implementação do software ou à documentação oficial do sistema ao qual o relatório de bug está associado. Esses links podem apontar para uma ou múltiplas páginas de documentação, incluindo guias, especificações, políticas internas, manuais técnicos ou

procedimentos formais. Na classificação manual, os comentários que possuem anexos deste tipo recebem uma etiqueta de 'Attachment' com 'AD'. O comentário ilustrado na Figura 24 foi classificado como *Documentation* por conter um link que direciona para a documentação oficial da Mozilla, descrevendo o processo de aprovação de bugs relacionados à segurança. Esse tipo de anexo fornece contexto normativo e orientações formais que auxiliam na correta condução do relatório.

Diferentemente da classe *Link*, que abrange referências externas de natureza geral, a classe *Documentation* é utilizada especificamente quando o link aponta para material documental estruturado, com caráter explicativo, normativo ou instrucional. Assim, essa distinção permite identificar comentários que fundamentam decisões ou esclarecimentos com base em documentação oficial. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955913 [17], e ilustra um padrão recorrente observado nos dados analisados, no qual referências à documentação técnica são utilizadas para orientar o tratamento adequado do relatório.



Figura 24: Comentário contendo um Attachment do tipo Documentation, retirado do relatório de bug ***ID: 955913*** *[17].*

Similar Bug Report

A classe *Similar Bug Report* abrange anexos representados por links que direcionam para outros relatórios de bugs que descrevem problemas semelhantes ou relacionados ao bug em análise. Esses links são utilizados para indicar possíveis duplicidades, dependências, casos correlatos ou exemplos prévios de comportamentos semelhantes

observados no sistema. Na classificação manual, os comentários que possuem anexos deste tipo recebem uma etiqueta de *Attachment* com 'ASB'. O comentário ilustrado na Figura 25 foi classificado como *Similar Bug Report* por conter um link que referência explicitamente outro relatório de bug, utilizado como exemplo de um problema semelhante ainda não resolvido ou previamente discutido.

Diferentemente da classe *Link*, que engloba referências externas de natureza geral, a classe *Similar Bug Report* é empregada especificamente quando o link aponta para outro relatório de bug dentro do próprio sistema de rastreamento, estabelecendo uma relação explícita entre casos. Essa distinção permite identificar conexões semânticas entre relatórios, que podem influenciar decisões de priorização, análise ou resolução. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955888 [16], e ilustra um padrão recorrente observado nos dados analisados, no qual referências a bugs similares são utilizadas para contextualizar o problema atual e enriquecer o processo de investigação.

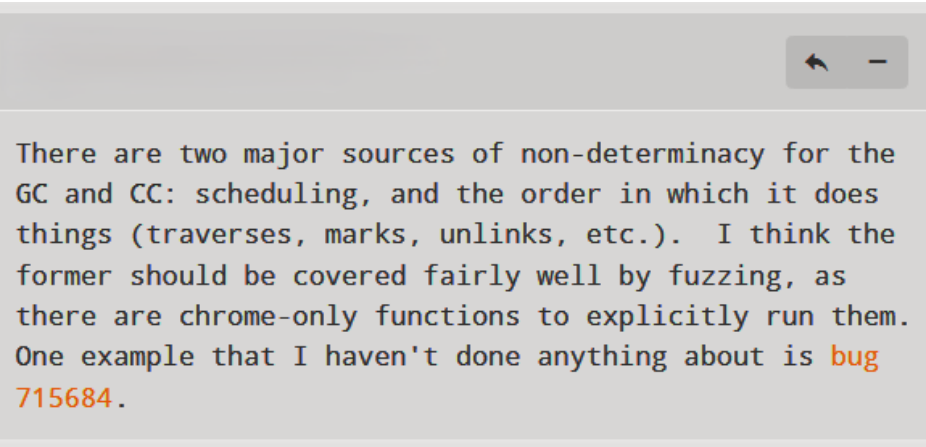


Figura 25: Comentário com Attachment do tipo Similar Bug Report, retirado do relatório de bug ***ID: 955888*** *[16].*

StackTrace

A classe *StackTrace* abrange anexos que apresentam pilhas de execução (*stack traces*) geradas durante a ocorrência do erro, seja no momento da reprodução do bug ou ao longo das tentativas de correção do problema relatado. Esses anexos descrevem a sequência de chamadas de funções ou métodos executados até o ponto de falha,

fornecendo informações técnicas detalhadas sobre o contexto de execução do sistema. Na classificação manual os comentários que possuem anexos deste tipo recebem uma etiqueta de *Attachment* com 'AS'. O comentário ilustrado na Figura 26 foi classificado como *StackTrace* por conter uma pilha de execução textual, indicando os métodos invocados e os caminhos de código percorridos até a ocorrência do erro.

Devido ao tamanho elevado que esse tipo de anexo pode assumir, a Figura 26 apresenta apenas um trecho representativo do conteúdo completo do comentário. Ainda assim, o exemplo ilustra um padrão recorrente observado nos dados analisados, no qual *stack traces* são utilizados como evidências técnicas fundamentais para a identificação da origem do problema e para o suporte às atividades de depuração. O exemplo apresentado foi extraído de um relatório de bug do sistema Bugzilla, identificado pelo ID 955913 [17], e evidencia o papel dos anexos do tipo *StackTrace* como recursos informativos de alto valor técnico no processo de análise e resolução de relatórios de bugs.

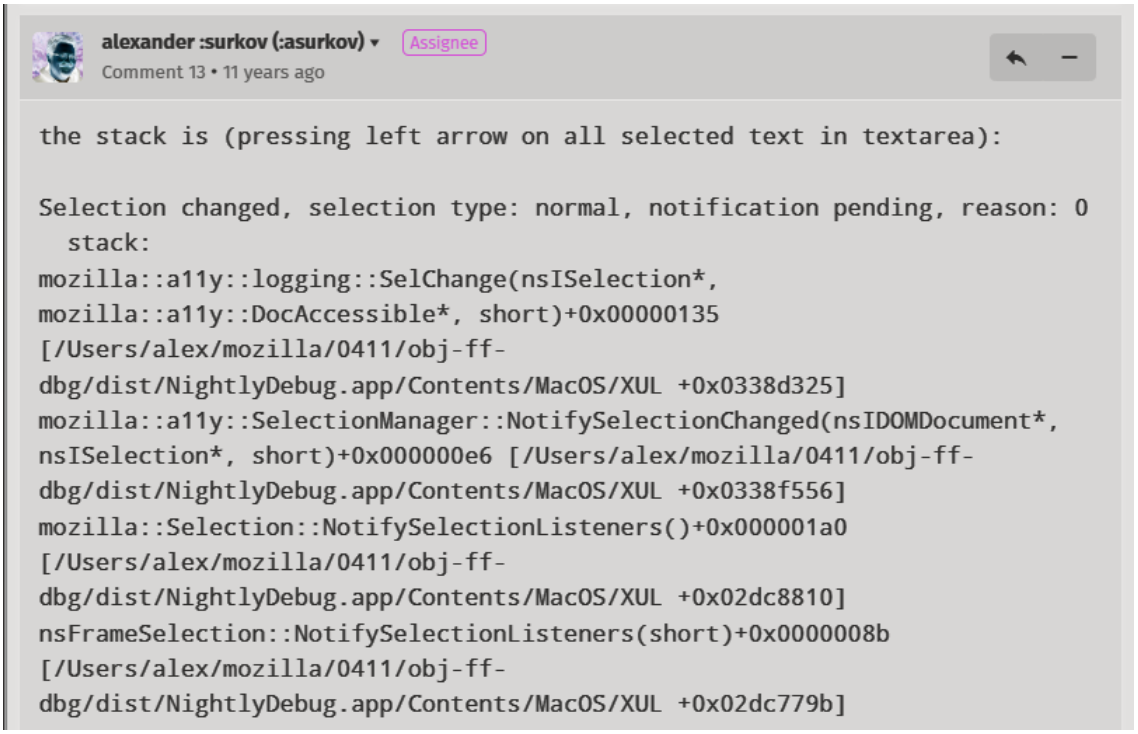


Figura 26: Comentário com Attachment do tipo StackTrace retirado do relatório de bug ***ID: 955913*** *[17].*

5.2 Considerações sobre a Definição e Aplicação das Classes de Comentários e Anexos

Durante a definição das classes de comentários e anexos, algumas questões conceituais surgiram, especialmente relacionadas à possibilidade de sobreposição entre classes, à atribuição de rótulos e à abrangência do modelo proposto. Este subcapítulo discute essas questões, com o objetivo de esclarecer as decisões adotadas neste estudo.

Embora um comentário possa, em alguns casos, apresentar características que se enquadrem em mais de uma subclasse da taxonomia, foi adotada a premissa de classificação única para o conteúdo textual. Assim, cada comentário é associado a apenas uma subclasse da hierarquia *Content Type*, correspondente àquele aspecto que melhor representa sua função predominante no contexto do relatório de bug. Essa decisão visa preservar a consistência do processo de classificação e evitar ambiguidades na utilização dos dados pelos modelos de *Machine Learning*.

No caso da hierarquia de *Attachment*, diferentemente do conteúdo textual, um comentário pode estar associado a zero, um ou múltiplos tipos de anexos simultaneamente. Essa distinção reflete a natureza complementar dos anexos, que podem coexistir em um mesmo comentário sem prejuízo à interpretação semântica do conteúdo textual principal. Dessa forma, enquanto a classificação de conteúdo é exclusiva, a classificação de anexos é cumulativa.

A taxonomia proposta foi construída de forma incremental a partir da análise empírica dos comentários presentes no conjunto de dados estudado. As classes e subclasses foram definidas com base nos padrões recorrentes observados, buscando abranger as formas mais representativas de contribuição textual e documental identificadas nos relatórios de bugs. Durante esse processo, não foram observados comentários que não pudessem ser adequadamente classificados em pelo menos uma das classes propostas, indicando que, no contexto deste estudo, a taxonomia se mostrou suficientemente abrangente. Cabe ressaltar que, embora a taxonomia seja considerada completa em relação ao conjunto de dados analisado, sua estrutura não é necessariamente exaustiva para todos os sistemas de rastreamento de bugs ou domínios

de software. Novas classes ou refinamentos podem ser incorporados em estudos futuros, à medida que diferentes contextos ou tipos de projetos sejam investigados.

A opção por uma classificação exclusiva do conteúdo textual foi adotada como uma simplificação metodológica, adequada aos objetivos deste estudo e ao uso de modelos supervisionados com rótulo único. No entanto, essa decisão implica uma limitação importante: comentários híbridos, que simultaneamente apresentem, por exemplo, elementos de discussão, questionamento e solução parcial, têm sua complexidade discursiva reduzida à classe considerada predominante. Assim, embora a estratégia de classificação única favoreça consistência e viabilidade experimental, ela não representa integralmente todos os casos de sobreposição semântica observáveis em comentários reais. A adoção de abordagens *multilabel* ou com sobreposição explícita entre classes constitui, portanto, uma possibilidade relevante para trabalhos futuros.

Por fim, a taxonomia proposta neste trabalho não surge em oposição a um conjunto consolidado de taxonomias equivalentes para comentários de relatórios de bugs, mas sim como uma ampliação de esforços ainda pontuais observados na literatura. Em particular, o estudo de Breu et al. [5] que realizou uma categorização voltada especificamente para perguntas presentes na seção de comentários, oferecendo uma contribuição relevante para compreender uma dimensão específica da interação. Entretanto, essa abordagem se restringe a comentários interrogativos e não contempla, de forma estruturada, outros papéis comunicativos recorrentes, como atualizações, diretrizes, soluções parciais, descrições técnicas e manifestações automatizadas. Nesse sentido, a taxonomia aqui proposta amplia esse recorte ao abranger diferentes funções comunicativas e técnicas observadas ao longo do ciclo de vida do relatório, permitindo uma análise mais abrangente da dinâmica colaborativa presente na seção de comentários.

6 Classificação Manual dos Comentários

Este capítulo apresenta os resultados da classificação manual dos comentários de relatórios de bugs, realizada com base na taxonomia de classes apresentada no capítulo 5 de Taxonomia. A classificação manual constitui um elemento central deste estudo, pois fornece um conjunto de dados rotulados que foi utilizado tanto para a análise descritiva dos comentários quanto como base para o treinamento e a avaliação dos modelos de aprendizagem de máquina empregados nas etapas subsequentes.

A classificação manual de comentários foi conduzida pelo autor principal deste trabalho, utilizando as classes definidas na taxonomia proposta. Ao longo do processo, as classificações atribuídas foram discutidas e validadas em reuniões de orientação do trabalho, com o objetivo de reduzir ambiguidades, garantir consistência conceitual entre as classes e mitigar possíveis vieses individuais. Para esse processo, foi utilizado um conjunto de dados previamente definido no subcapítulo 4.3 de caracterização de dados, composto por 10.000 relatórios de bugs e um total de 100.046 comentários associados a esses relatórios. Dentre esses relatórios, 1069 relatórios de bugs tiveram todos os seus comentários analisados e classificados manualmente, resultando em um total de 37.862 comentários classificados.

A seleção desses 1069 relatórios foi realizada por meio de amostragem aleatória simples a partir do conjunto total de relatórios descrito na Seção 4.3. A unidade de seleção foi o relatório de bug, e, uma vez selecionado, todos os comentários associados a esse relatório foram integralmente classificados manualmente. Essa estratégia preserva a estrutura completa das threads de discussão e evita a fragmentação do contexto conversacional durante o processo de anotação. A Figura 27 apresenta uma visualização simplificada da estrutura de um comentário após o processo de classificação manual. No exemplo ilustrado, o comentário foi rotulado com *Content Type*: ‘D’, indicando sua classificação na classe *Discussion*, e com *Attachment*: *‘AL’*, denotando a presença de um anexo do tipo *Link*.



Figura 27: Visualização simplificada de um comentário rotulado.

A meta estabelecida neste estudo foi classificar manualmente pelo menos 20% do total aproximado de 100.000 comentários disponíveis no conjunto de dados. Esse percentual foi escolhido como um compromisso entre a viabilidade do esforço manual de rotulação e a necessidade de obter um volume suficientemente representativo de dados anotados para o treinamento supervisionado dos modelos de machine learning. A meta foi alcançada, uma vez que os 37.862 comentários classificados correspondem a aproximadamente 37,8% do total de comentários disponíveis. Os comentários classificados manualmente foram utilizados como base para o treinamento e a validação dos modelos de classificação automática de comentários de relatórios de bugs, permitindo avaliar a capacidade dos algoritmos de *Machine Learning* em reproduzir os rótulos atribuídos manualmente e apoiar análises posteriores relacionadas ao tempo de resolução e à evolução dos relatórios.

6.1 Análises sobre os Comentários classificados manualmente

Com a etapa de classificação manual de comentários e anexos, foi possível consolidar um panorama detalhado sobre a distribuição das classes de comentários presentes em

relatórios de bugs. Ao todo, foram rotulados 37862 comentários que foram distribuídos em dez classes distintas de comentários. A Tabela 1 apresenta a quantidade total de comentários classificados em cada uma das classes. Os dados mostram que as classes *Bot Comment* e *Description* são as mais numerosas com, respectivamente, 14971 e 9622 comentários. Esse resultado está diretamente relacionado à própria estrutura da plataforma Bugzilla e ao uso de ferramentas automatizadas no processo de desenvolvimento. Em particular, todo relatório de bug no Bugzilla possui obrigatoriamente um comentário inicial (comentário 0), registrado no momento da criação do relatório, que contém a descrição original do problema fornecida pelo autor.

Por definição da plataforma, esse comentário não representa uma interação posterior, sendo classificado diretamente como *Description*. Já os comentários do tipo *Bot Comment* foram identificados a partir da recorrência de mensagens geradas automaticamente por agentes específicos da plataforma, como bots de integração contínua e execução de testes. Esses comentários apresentam padrões textuais e autoria consistentes, permitindo sua identificação sistemática e rotulação como comentários automáticos.

Tabela 1: Quantidade de comentários classificados por classe de comentários da Taxonomia

Classes	**Quantidade de comentários**
Discussion	1067
Update	3640
Error	203
No Text	6869
Description	9622
Partial Solution With Code	267
Partial Solution Without Code	96
Question	455
Guideline	672
Bot Comment	14971

Os resultados demonstram que os tipos de comentários não ocorrem de forma equilibrada em relatórios de bugs. Em plataformas como o Bugzilla, é natural que alguns tipos de comentários, como descrições, comentários automáticos de bots e atualizações apareçam com mais frequência do que comentários de caráter mais técnicos ou colaborativos. Essa distribuição naturalmente desbalanceada reflete o comportamento real dos desenvolvedores e das ferramentas que são utilizadas nos ambientes de desenvolvimento de software. Assumindo a corretude da classificação manual realizada neste estudo, ainda que sujeita a interpretações e decisões humanas inerentes a processos de anotação manual, foi conduzida uma etapa adicional de balanceamento dos dados com o objetivo de reduzir riscos de enviesamento nas etapas posteriores de classificação automática por meio de algoritmos de aprendizagem de máquina. Esse balanceamento foi aplicado exclusivamente após a conclusão da classificação manual.

Nessa etapa, as classes majoritárias *Bot Comment*, *Description* e *No Text* tiveram sua quantidade limitada a 5390 comentários por classe, por meio de

amostragem controlada. Essa limitação não alterou a rotulação original dos comentários, mas definiu quais instâncias seriam utilizadas no conjunto de treinamento dos modelos.

Essa estratégia permitiu a construção de um conjunto de dados mais equilibrado, assegurando que classes menos frequentes tivessem representação adequada durante o processo de aprendizado dos modelos de *Machine Learning*. A Tabela 2 apresenta as quantidades finais de comentários por classe utilizadas nas etapas de classificação automática.

Tabela 2: Quantidade de comentários classificados por classe da taxonomia após o ajuste de balanceamento do conjunto de dados.

Classes	**Quantidade de comentários**
Discussion	1067
Update	3640
Error	203
No Text	5390
Description	5390
Partial Solution With Code	267
Partial Solution Without Code	96
Question	455
Guideline	672
Bot Comment	5390

Tabela 3: Quantidade de anexos presentes em comentários

Conceito - Attachment	Quantidade de comentários
Link	5827
File	554
Similar Bug Report	2569
Stacktrace	2
Documentation	18
No attachments	13598

De modo geral, esses resultados obtidos indicam que a comunicação nos relatórios de bugs é caracterizada por uma combinação entre discussões técnicas, mensagens automáticas, atualizações informativas e referências externas. Essa diversidade de interações reflete uma complexidade colaborativa no processo de manutenção de software e serviu como base para as etapas seguintes de experimentos do estudo, que buscam compreender como cada tipo de comentário e anexo pode influenciar no tempo e na eficácia de resolução de bugs.

6.2 Análises da Distribuição das Classes de Comentários e Anexos

A classe *Update*, com 3.640 comentários na Tabela 2, apresenta a maior frequência entre as classes analisadas, evidenciando que grande parte das interações nos relatórios de bugs está relacionada à atualização de informações, mudanças de status e complementação de dados ao longo do ciclo de vida do relatório. Esse resultado indica que os relatórios de bugs são artefatos dinâmicos, frequentemente atualizados à medida que o problema evolui até sua resolução.

Em seguida, a classe *Discussion*, com 1.067 comentários, reflete a presença de interações colaborativas entre desenvolvedores, caracterizadas por trocas de informações, esclarecimentos e debates técnicos durante o processo de análise e

correção dos problemas reportados. A ocorrência significativa dessa classe reforça o papel dos comentários como um canal de comunicação ativa dentro das comunidades de desenvolvimento.

As classes *Guideline* e *Question*, com respectivamente 672 e 455 comentários, apresentam frequências intermediárias. Comentários do tipo *Guideline* estão associados à disponibilização de orientações técnicas e referências que auxiliam na resolução do problema, enquanto os comentários classificados como *Question* representam dúvidas ou solicitações de esclarecimentos direcionadas aos envolvidos no relatório. Essas classes indicam momentos de orientação e investigação ao longo do processo de resolução do bug.

As classes *Partial Solution With Code* e *Partial Solution Without Code*, com 267 e 96 comentários respectivamente, revelam contribuições diretas, ainda que incompletas, para a resolução dos problemas. A menor frequência dessas classes sugere que propostas de soluções parciais, especialmente aquelas acompanhadas de código, são menos comuns do que discussões ou atualizações, mas desempenham um papel técnico relevante ao impulsionar o avanço na correção dos bugs. Por fim, a classe *Error*, com 203 comentários, concentra descrições de falhas, exceções ou comportamentos incorretos observados durante a replicação ou tentativa de correção dos problemas.

No que diz respeito aos anexos associados aos comentários, os resultados apresentados na Tabela 3 mostram que aproximadamente 60% dos comentários (13.598) não possuem qualquer tipo de anexo. Entre os comentários que apresentam anexos, o tipo mais frequente é *Link*, com 5.827 ocorrências, seguido por *Similar Bug Report*, com 2.569 comentários, e *File*, com 554 comentários. Os tipos *StackTrace* e *Documentation* aparecem de forma pontual, com apenas 2 e 18 comentários, respectivamente.

Essa distribuição indica que, embora a maioria dos comentários não contenha anexos estruturais, quando presentes, os anexos registrados formalmente na plataforma são majoritariamente do tipo *Link*. Essa constatação decorre da classificação baseada no tipo de recurso identificado via "*attachment_id*", e não da análise semântica detalhada do conteúdo de cada link. Assim, a predominância de links sugere um padrão de referência a recursos externos ou a outros relatórios, conforme categorizado na

taxonomia adotada. Esse comportamento sugere que desenvolvedores frequentemente recorrem a fontes externas e a problemas semelhantes como apoio ao processo de resolução, enquanto anexos mais estruturados, como *stacktraces* completos ou documentação formal, são menos comuns.

De modo geral, os resultados obtidos indicam que a comunicação nos relatórios de bugs é marcada por uma combinação de atualizações informativas, interações colaborativas, contribuições técnicas parciais e referências externas. Essa diversidade de interações evidencia a complexidade do processo colaborativo de manutenção de software e fundamenta as etapas subsequentes deste estudo, que buscam analisar como os diferentes tipos de comentários e anexos influenciam o tempo e a eficácia da resolução de relatórios de bugs.

Embora esta etapa esteja vinculada à QP1, ao caracterizar quantitativamente a distribuição das classes de comentários, ela também estabelece a base para a investigação proposta na QP2. Ao compreender quais tipos de comentários são mais frequentes e como se distribuem nos relatórios analisados, torna-se possível examinar posteriormente se determinados perfis comunicacionais estão associados a diferentes padrões de duração de resolução.

7 Resultados do Experimento de classificação de comentários

Esse capítulo trata dos resultados gerais e específicos para cada um dos modelos utilizados na etapa de experimento de classificação automática de comentários. Para os modelos supervisionados, serão apresentados os resultados obtidos com *Naive Bayes*, *CNN*, *Decision Tree*, *SVM*, *Random Forest* e *Logistic Regression*. Para os modelos não supervisionados, serão apresentados os resultados obtidos com *DBSCAN*, *HDBSCAN* e *K-Means*. Por fim, para as LLMs, serão apresentados os resultados obtidos com *Gemini* e *T5*.

Esta seção apresenta os resultados do experimento de classificação automática dos comentários e está diretamente relacionada à QP1, que investiga como os comentários de relatórios de bugs podem ser classificados considerando seu conteúdo textual, a presença de anexos e seu papel no processo de resolução. Os resultados a seguir avaliam a viabilidade dessa classificação automática por meio de diferentes abordagens de machine learning, permitindo verificar empiricamente a consistência e os limites da taxonomia proposta.

7.1 Modelos Supervisionados

Nesse subcapítulo iremos demonstrar os resultados de classificação realizada por cada um dos modelos primeiramente de forma separada, e, depois, mostraremos comparativos entre os melhores cenários de cada modelo. Para tal, dividimos a apresentação dos resultados em duas fases: a primeira onde usamos as métricas de Acurácia, F1-Score, Revocação e Precisão para cada um dos modelos supervisionados em seus cenários executados no modelo; a segunda, na qual é analisado como o desempenho global dos modelos é influenciado pelo comportamento individual de cada classe de comentários, por meio da avaliação das métricas por classe nos cenários de melhor e pior desempenho de cada modelo.

No decorrer deste subcapítulo, utilizamos o termo cenário para designar a combinação entre o conjunto de dados empregado e a configuração de hiperparâmetros do modelo. Cada cenário é composto, portanto, por um tipo de *dataset* desbalanceado (D) ou ajustado por técnicas de reamostragem, como *Oversampling* (O), *Undersampling* (U) e *SMOTE* (S); associado à execução do modelo sem ajuste fino de hiperparâmetros (NF) ou com ajuste fino de hiperparâmetros (F). Mais detalhes sobre as variações com técnicas de reamostragem podem ser encontrados no subcapítulo 4.7 de Variações de Dados.

Nos cenários com ajuste fino (F), a configuração de hiperparâmetros utilizada corresponde à melhor combinação encontrada pelo processo de *GridSearch* [56], conforme descrito no subcapítulo 4.9. Assim, cada cenário com fine-tuning representa o melhor desempenho obtido para aquele *dataset*, e não diferentes execuções para cada combinação individual de hiperparâmetros avaliadas durante a busca. Também são reportadas Macro-métricas [59] (F1 Macro, Revocação (*recall*) macro e precisão macro), calculadas como a média entre as métricas de cada classe, atribuindo o mesmo peso a todas elas. Essas métricas são especialmente relevantes em dados desbalanceados, pois destacam o comportamento do modelo nas classes minoritárias, mesmo quando a acurácia global e o *F1 Weighted* permanecem elevados.

Naive Bayes

O modelo *Naive Bayes* foi avaliado para cada um dos oito cenários desenvolvidos no experimento. Por cenário, consideramos a combinação do *dataset* de comentários desbalanceado, do *dataset* ajustado com *Oversampling*, *Undersampling* e *Smote* junto com os ajustes finos de hiperparâmetros.

A tabela 4 mostra os valores de todas as métricas coletadas em cada cenário executado. De modo geral, os resultados revelam um padrão consistente em que o modelo mantém valores de acurácia relativamente altos, com variação de 0.78 a 0.84 dependendo do cenário, e *F1 Weighted* elevado (aproximadamente de 0.80). Contudo, o modelo apresentou valores inferiores nas macro-métricas, o que evidencia o impacto do desbalanceamento natural do *dataset*: o desempenho médio entre as classes,

especialmente as minoritárias, é significativamente menor do que o desempenho global sugerido pela acurácia e pelo *F1 Weighted*.

Tabela 4: Métricas dos cenários do modelo Naive Bayes

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.83	0.37	0.38	0.48	0.78
D-F	0.84	0.50	0.47	0.58	0.82
O-NF	0.78	0.50	0.57	0.49	0.81
O-F	0.80	0.50	0.53	0.49	0.82
U-NF	0.70	0.44	0.54	0.45	0.74
U-F	0.72	0.45	0.53	0.46	0.76
S-NF	0.78	0.50	0.57	0.49	0.81
S-F	0.80	0.50	0.54	0.49	0.82

O fine-tuning do modelo *Naive Bayes*, em especial o ajuste do parâmetro de suavização (*alpha*), apresentou impactos consistentes e positivos. Neste estudo, um cenário "com fine-tuning" (F) representa o melhor resultado obtido para aquele dataset após o *GridSearch*, isto é, a combinação do dataset com a configuração de hiperparâmetros que maximiza o desempenho no conjunto de validação. No cenário D-NF (Desbalanceado sem fine-tuning), o modelo obteve F1 Macro de 0.37. Com a aplicação do fine-tuning no cenário D-F, esse valor aumentou para 0.50, indicando melhora no desempenho médio entre as classes, demonstrando assim que o modelo respondeu bem à calibragem adequada. Nos cenários de *Oversampling*, *Undersampling* e *SMOTE*, os ajustes de hiperparâmetros mantiveram o comportamento estável, com pequenas variações, mas sempre contribuindo para um equilíbrio melhor entre precisão e revocação para as classes minoritárias.

Os cenários executados no experimento com *Naive Bayes* mostraram alguns padrões: *Oversampling* e *SMOTE* apresentaram resultados semelhantes, ambos alcançando os maiores valores de revocação macro (0.57) entre todos os outros cenários, evidenciando assim que a ampliação do número de exemplos das classes minoritárias aumenta efetivamente a sensibilidade do modelo para essas classes; O cenário O-F (*Oversampling* com fine-tuning) manteve o F1 Macro estável em 0.50, mas compensou parte da queda na acurácia observada no O-NF (*Oversampling* sem fine-tuning); O *Undersampling* levou à maior queda de desempenho geral, com acurácia na faixa de 0.70 a 0.72 e F1 Macro entre 0.44 e 0.45, demonstrando que reduzir drasticamente as classes majoritárias prejudica o poder discriminativo do modelo; Por fim, observamos que o *Oversampling* e o *SMOTE* embora elevem a Revocação macro, eles também reduzem a acurácia refletindo um *trade-off* entre o desempenho global e a sensibilidade às classes menos representadas.

O melhor cenário global para o *Naive Bayes* foi o cenário D-F (Desbalanceado com fine-tuning), que obteve a maior acurácia (0.84) combinada ao maior F1 Macro observado (0.50). Nesse cenário, o *GridSearch* selecionou alpha = 0.01 como melhor hiperparâmetro, resultando em acurácia de (0.84) combinada ao maior F1 Macro observado (0.50), mostrando que os ajustes de hiperparâmetros compensaram parcialmente as distorções provocadas pelo desbalanceamento. Os piores resultados ocorrem com os cenários de *Undersampling*, que deterioram a acurácia e prejudicam tanto a precisão quanto o F1 das classes majoritárias.

Decision Tree

A Tabela 5 mostra os valores de todas as métricas coletadas em cada cenário executado. De forma geral, o modelo *Decision Tree* apresentou métricas de acurácia variando entre valores de 0.67 e 0.83; para a F1 Macro com valores entre 0.42 e 0.56 a depender do cenário. As métricas *F1 weighted* permanecem elevadas (com valores entre 0.81 e 0.83 na maioria dos cenários), enquanto as macro-métricas indicam uma desigualdade significativa entre as classes de comentários, em especial das classes minoritárias.

Tabela 5: Métricas dos cenários do modelo Decision Tree

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.83	0.52	0.50	0.54	0.82
D-F	0.81	0.52	0.54	0.50	0.82
O-NF	0.81	0.52	0.54	0.51	0.82
O-F	0.82	0.54	0.56	0.53	0.83
U-NF	0.70	0.43	0.49	0.44	0.75
U-F	0.67	0.42	0.53	0.44	0.71
S-NF	0.80	0.50	0.52	0.49	0.81
S-F	0.80	0.50	0.52	0.49	0.81

O comportamento do modelo *Decision Tree* nos diferentes cenários evidencia a sensibilidade quanto ao desbalanceamento e a distribuição das classes de comentários. No cenário desbalanceado sem fine-tuning (D-NF), o modelo apresentou uma acurácia elevada (0.83), mas com valores mais baixos em F1 Macro (0.52) e Revocação Macro (0.50), indicando com isso uma forte dependência das classes majoritárias. Com os ajustes de hiperparâmetros (D-F), ficou notável um aumento na Revocação macro (elevação da métrica para 0.54) o que sugere uma maior capacidade de recuperação de instâncias das classes minoritárias embora que a custa de uma leve queda na acurácia (0.81).

Nos cenários de *Oversampling*, o modelo manteve desempenho consistente, e a combinação com fine-tuning (O-F) atingiu o melhor equilíbrio geral entre o desempenho global e as métricas macro. Nesse cenário, observa-se o maior valor de F1 Macro (0.54) e a maior Revocação Macro (0.56), além de uma acurácia também alta (0.82). Esses dados evidenciam que aumentar artificialmente a representação das classes minoritárias, somado ao ajuste fino do modelo de *Decision Tree*, melhora a sensibilidade entre as classes sem comprometer excessivamente a generalização.

Os cenários com *Undersampling* apresentaram os piores resultados do modelo de *Decision Tree*, com valores de acurácia reduzidos (0.67 a 0.70) e F1 Macro abaixo de 0.45. Esses resultados revelaram que a remoção de grandes quantidades de instâncias das classes majoritárias prejudicou significativamente a estrutura de decisão do modelo de *Decision Tree*, reduzindo sua capacidade de discriminação. Ainda assim, observa-se um aumento do Revocação macro no cenário de *Undersampling* com fine-tuning (U-F), atingindo o valor de (0.53), indicando maior sensibilidade às classes minoritárias, embora com prejuízo ao desempenho global.Em resumo, o melhor cenário encontrado para o modelo *Decision Tree* foi o cenário O-F (*Oversampling* com fine-tuning), que apresentou uma das maiores acurácias (0.82) aliada aos maiores valores de F1 Macro (0.54) e Revocação macro (0.56). Nesse cenário, o processo de *GridSearch* selecionou os seguintes hiperparâmetros: (*criterion = entropy, max_depth = None, min_samples_split = 2, min_samples_leaf = 1 e class_weight = None*). Esses resultados indicam que a combinação entre redistribuição das classes e uma estrutura de árvore não limitada em profundidade favoreceu a captura de padrões relevantes das classes minoritárias, sem comprometer significativamente a capacidade de generalização do modelo. Os piores desempenhos foram observados nos cenários de *Undersampling*, nos quais a remoção agressiva de instâncias das classes majoritárias comprometeu a estrutura de decisão da árvore, resultando em quedas significativas tanto na acurácia quanto nas métricas macros.

Random Forest

A Tabela 6 mostra os valores de todas as métricas coletadas em cada cenário executado. De forma geral, o modelo *Random Forest* apresentou comportamentos consistentes, especialmente nos cenários desbalanceados e com *Oversampling*. A acurácia variou de valores entre 0.71 e 0.86, com valores de *F1 Weighted* consistentemente elevados (entre 0.75 e 0.86) indicando forte capacidade de modelar corretamente as classes majoritárias. Porém, as macrométricas permaneceram moderadas refletindo que o modelo enfrenta desafios na identificação das classes menos representadas.

Tabela 6: Métricas dos cenários do modelo Random Forest

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.85	0.51	0.48	0.68	0.83
D-F	0.86	0.55	0.53	0.60	0.85
O-NF	0.86	0.56	0.55	0.60	0.85
O-F	0.86	0.55	0.53	0.59	0.85
U-NF	0.71	0.46	0.55	0.48	0.75
U-F	0.72	0.47	0.56	0.49	0.76
S-NF	0.85	0.55	0.54	0.58	0.85
S-F	0.86	0.57	0.56	0.59	0.85

Os resultados dos cenários do modelo *Random Forest* o caracterizam como um modelo robusto e adaptável que consegue manter o desempenho elevado mesmo em um dataset de dados altamente desbalanceado. No cenário D-NF (Desbalanceado sem fine-tuning) o modelo já apresenta uma acurácia de 0.85 e *F1 Weighted* de 0.83, evidenciando uma boa generalização sobre as classes majoritárias do dataset. Após a aplicação do fine-tuning (D-F) podemos observar que houve um incremento relevante nas macro-métricas F1 Macro subindo de 0.51 para 0.55, e a Revocação Macro de 0.48 para 0.53, demonstrando maior sensibilidade às classes minoritárias sem comprometer a acurácia.

Os cenários com *Oversampling* apresentaram desempenho estável tanto com fine-tuning quanto sem fine-tuning (O-F e O-NF), em ambos a acurácia permaneceu em 0.86, enquanto as macro-métricas variaram entre valores de 0.55 e 0.56. Essa estabilidade sugere que o *Random Forest* se beneficia da redistribuição artificial das classes de comentários elevando a capacidade de discriminação sobre classes pouco representadas sem prejudicar a performance global.

Nos cenários com *Undersampling* observaram-se desempenhos inferiores. A acurácia caiu para a faixa de 0.71 a 0.72 e o F1 Macro sofreu uma queda acentuada, permanecendo abaixo de 0.47. Embora a Revocação Macro atinja valores maiores nesses cenários com valores de 0.55 a 0.56, isso ocorre à custa de perda de precisão, indicando um aumento de falsos positivos nas classes minoritárias, demonstrando um reflexo direto da remoção brusca de exemplos das classes dominantes. No modelo *Random Forest*, o melhor cenário geral foi o S-F (SMOTE com fine-tuning), que apresentou uma combinação equilibrada das métricas de avaliação, alcançando Acurácia de 0.86, F1 Macro de 0.57 e Revocação macro de 0.56. Nesse cenário, o processo de *GridSearch* selecionou os seguintes hiperparâmetros: *(n_estimators = 200, max_depth = None, min_samples_split = 5, min_samples_leaf = 1 e class_weight = None)*. Esses resultados indicam que o aumento do número de árvores aliado a uma profundidade não restrita permitiu ao modelo capturar padrões mais complexos das classes minoritárias, especialmente quando combinado com a técnica de *SMOTE*.

Os piores resultados foram observados nos cenários de *Undersampling* (U-NF e U-F), nos quais a redução agressiva das instâncias das classes majoritárias comprometeu tanto a acurácia quanto a precisão e o F1 Macro. Embora a Revocação macro tenha apresentado aumento nesses cenários, o custo em termos de falsos positivos tornou essas configurações menos adequadas para uma classificação equilibrada dos comentários.

Support Vector Machine (SVM)

A Tabela 7 mostra os valores de todas as métricas coletadas em cada cenário executado. De forma geral, o *Support Vector Machine* apresentou bons desempenhos globais, com valores de acurácia variando de 0.74 a 0.86, que se trata de um modelo robusto frente ao desbalanceamento natural do conjunto de dados de comentários. Porém, os resultados das macro-métricas: Precisão macro, Revocação macro e o F1 Macro mostram que o modelo possui pontos fracos, em especial na discriminação das classes minoritárias.

Tabela 7: Métricas dos cenários do modelo Support Vector Machine

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.86	0.54	0.51	0.72	0.84
D-F	0.86	0.57	0.57	0.59	0.85
O-NF	0.84	0.56	0.57	0.56	0.85
O-F	0.86	0.56	0.54	0.60	0.85
U-NF	0.75	0.49	0.58	0.49	0.79
U-F	0.75	0.50	0.57	0.50	0.79
S-NF	0.84	0.55	0.57	0.55	0.85
S-F	0.86	0.55	0.53	0.60	0.85

Os resultados do *Support Vector Machine* (SVM) mostram um comportamento sólido nos cenários executados sem reduções agressivas por conta das diferenças do dataset. No cenário D-NF, o modelo apresentou uma acurácia de 0.86 e um *F1 Weighted* de valor 0.84, com F1 Macro de 0.54, esses valores indicam uma boa capacidade de separação entre as classes principais embora ainda tenha demonstrado um desempenho limitado nas classes minoritárias. Ao aplicar ajustes de hiperparâmetros (D-F), o modelo apresentou aumento no F1 Macro para 0.57, mantendo os valores de acurácia e *F1 Weighted* estáveis sugerindo que os ajustes contribuíram para melhorar o equilíbrio entre sensibilidade e especificidade no modelo.

Nos cenários com *Oversampling* (O-NF e O-F), o *Support Vector Machine* (SVM) também se manteve com desempenho estável, com acurácia com valores entre 0.84 e 0.86 e F1 Macro em 0.56. O *Oversampling* aumentou a sensibilidade para classes minoritárias sem introduzir instabilidade significativas, mantendo um *F1 Weighted* consistente com valor de 0.85. Quanto ao ajuste de hiperparâmetros para esse cenário de dados, o O-F apresentou valores próximos ao melhor desempenho global do SVM, mostrando uma robustez nesse tipo de ajuste.

Os cenários com *Undersampling* (U-NF e U-F) apresentaram os piores resultados do SVM, com valores de acurácia de 0.75 e F1 Macro entre 0.49 e 0.50. Essa degradação nos valores pode ser explicada pela perda substancial de exemplos nas classes majoritárias, prejudicando com isso o espaço de separação de margens, técnica de funcionamento do SVM, e compromete assim o desempenho geral quanto a capacidade de distinguir adequadamente as classes minoritárias. Por fim, os cenários com *SMOTE* (S-NF e S-F) apresentaram comportamentos similares aos cenários de *Oversampling* porém mostrando ligeira vantagem em algumas métricas. O cenário S-F atingiu uma acurácia de 0.86 enquanto o *F1 Weighted* com valor de 0.85 e F1 Macro de 0.55, mantendo uma distribuição equilibrada entre as macro-métricas e sugerindo que a interpolação sintética de exemplos ajudou a suavizar a fronteira de decisão para a discriminação com as classes minoritárias. O melhor cenário global para o *Support Vector Machine* (SVM) foi o D-F (Desbalanceado com fine-tuning), que obteve acurácia de 0.86 e F1 Macro de 0.57, representando o maior valor dessa métrica entre todos os cenários avaliados. Nesse cenário, o processo de *GridSearch* selecionou os seguintes hiperparâmetros: *(kernel = rbf, C = 1, gamma = scale, degree = 2 e class_weight = balanced)*. A utilização de pesos balanceados permitiu ao modelo compensar parcialmente o desbalanceamento natural do conjunto de dados, enquanto o "*kernel = RBF"* favoreceu a construção de fronteiras de decisão mais flexíveis, resultando em melhor discriminação das classes minoritárias sem comprometer o desempenho global.

Os piores desempenhos foram observados nos cenários de *Undersampling* (U-NF e U-F), nos quais a redução excessiva do conjunto de dados prejudicou a definição adequada das margens de separação, levando a quedas consistentes tanto na acurácia quanto no F1 Macro.

Logistic Regression

A Tabela 8 mostra os valores de todas as métricas coletadas em cada cenário executado. O modelo apresentou um desempenho global sólido, com valores de acurácia variando entre 0.75 e 0.86 e *F1 Weighted* entre 0.79 e 0.86 que demonstram uma boa capacidade de modelar relações lineares nos dados textuais dos comentários.

Tabela 8: Métricas dos cenários do modelo Logistic Regression

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.86	0.51	0.48	0.68	0.83
D-F	0.84	0.57	0.58	0.56	0.85
O-NF	0.83	0.56	0.59	0.54	0.84
O-F	0.82	0.53	0.55	0.52	0.83
U-NF	0.75	0.49	0.56	0.48	0.79
U-F	0.84	0.57	0.57	0.56	0.85
S-NF	0.83	0.56	0.60	0.54	0.85
S-F	0.82	0.51	0.54	0.50	0.82

No cenário desbalanceado sem fine-tuning (D-NF) o modelo alcançou valor em acurácia de 0.86, *F1 Weighted* de 0.83 e F1 Macro de 0.51, valores esses que se alinham ao desempenho dos melhores modelos lineares participantes deste experimento. O ajuste de hiperparâmetros (D-F) aumentou o F1 Macro para 0.57 embora tenha ocorrido uma pequena queda no valor da acurácia mostrando que a calibragem dos hiperparâmetros contribuiu com uma melhora na sensibilidade com as classes minoritárias sem deteriorar significativamente o desempenho geral do cenário.

Nos cenários com *Oversampling* (O-NF e O-F) o modelo manteve comportamento estável com valores de acurácia entre 0.82 e 0.83, e F1 Macro entre 0.53 e 0.56. Mesmo o *Oversampling* ajudando a reduzir a disparidade entre as classes de comentários, o modelo ainda apresentou maior precisão nas classes majoritárias e preservou o valor de *F1 Weighted* em torno de 0.84, indicando boa capacidade de generalização.

Os cenários com *Undersampling* (U-NF e U-F) mostraram resultados mais variáveis se comparados aos outros cenários. No cenário U-NF a acurácia caiu para 0.75 e F1 Macro para 0.49, refletindo uma perda significativa de informação causada pela redução das classes majoritárias. O cenário U-F recuperou parte da perda atingindo uma

acurácia de 0.84 e F1 Macro de 0.57 mostrando um efeito de melhora através do ajuste fino de hiperparâmetros, que compensou parcialmente a perda de dados e melhorou o equilíbrio entre as classes.

Por fim, os cenários com *SMOTE* (S-NF e S-F) apresentaram desempenhos consistentes com os cenários que utilizaram *Oversampling*. O cenário S-NF atingiu um valor de acurácia de 0.83 e F1 Macro de 0.56, demonstrando que a interpolação sintética favoreceu o aprendizado das classes minoritárias sem comprometer a estabilidade do modelo. Os ajustes de hiperparâmetros no cenário S-F não trouxeram ganhos significativos, mantendo os resultados próximos aos obtidos sem uso do fine-tuning.

O melhor cenário global para o modelo *Logistic Regression* foi o D-F (Desbalanceado com fine-tuning), que alcançou um valor de F1 Macro de 0.57, o maior dentre os cenários avaliados para esse modelo, combinado a um *F1 Weighted* de 0.85. Nesse cenário, o processo de *GridSearch* selecionou os seguintes hiperparâmetros: *(C = 1, penalty = l2, solver = liblinear e class_weight = balanced)*. A utilização de regularização L2 aliada ao balanceamento de pesos das classes contribuiu para uma melhor separação linear entre as classes de comentários, aumentando a sensibilidade às classes minoritárias sem comprometer o desempenho global do modelo.

O pior desempenho foi observado no cenário U-NF (*Undersampling* sem fine-tuning), no qual a redução excessiva de instâncias das classes majoritárias resultou em degradação consistente das métricas, especialmente da acurácia (0.75) e do F1 Macro (0.49).

Convolutional Neural Network (CNN)

A Tabela 9 mostra os valores de todas as métricas coletadas em cada cenário executado. Diferentemente dos demais modelos avaliados neste experimento, a *CNN* demonstrou grande sensibilidade ao desbalanceamento e a quantidade reduzida de dados por classe, resultando assim em um desempenho substancialmente inferior em todos os cenários, principalmente se comparados aos cenários dos outros modelos utilizados no experimento.

Os valores de acurácia variaram entre 0.24 e 0.54 enquanto o *F1 Weighted* permaneceu entre 0.18 e 0.54, valores esses significativamente abaixo dos observados

nos outros modelos. As macro-métricas foram igualmente baixas, com F1 Macro variando entre 0.08 e 0.28, refletindo uma dificuldade extrema do modelo em capturar padrões tanto nas classes majoritárias como nas classes minoritárias.

Tabela 9: Métricas dos cenários do modelo Convolutional Neural Network

Cenário	Acurácia	F1 Macro	Revocação macro	Precisão macro	*F1 Weighted*
D-NF	0.52	0.20	0.24	0.23	0.43
D-F	0.55	0.22	0.25	0.22	0.48
O-NF	0.39	0.20	0.26	0.22	0.40
O-F	0.45	0.22	0.28	0.27	0.44
U-NF	0.28	0.09	0.17	0.09	0.19
U-F	0.33	0.15	0.23	0.14	0.30
S-NF	0.46	0.23	0.27	0.27	0.47
S-F	0.48	0.23	0.28	0.35	0.49

Os resultados mostraram que o modelo *CNN* apresentou dificuldade consistente em todos os cenários indicando que o modelo não conseguiu aprender representações adequadas dos textos dos comentários mesmo com as técnicas de balanceamento do dataset e com os ajustes de hiperparâmetros. O melhor cenário ocorreu em D-F (Desbalanceado com fine-tuning) com valores de acurácia de 0.55, *F1 Weighted* com 0.48 e F1 Macro de 0.22. Desempenho esse longe dos apresentados nos outros modelos participantes do experimento de classificação multiclasse.

Os cenários com *Oversampling* (O-NF e O-F) trouxeram pioras na estabilidade das previsões, com valores de acurácia baixos (entre 0.39 e 0.45) e F1 Macro em 0.22. O *Undersampling* apresentou os piores resultados gerais, com acurácia de 0.28 e *F1 Weighted* de 0.19, mostrando com isso que a redução do dataset agravou ainda mais a limitação da rede em aprender padrões textuais.

Os cenários com *SMOTE* (S-NF e S-F) também não apresentaram ganhos significativos embora tenham sido ligeiramente superiores aos cenários com undersampling. A acurácia máxima atingida pelo modelo nos cenários com *SMOTE* foi de 0.48 e F1 Macro de 0.23, valores esses que indicam que o modelo *CNN* não conseguiu modelar a distribuição das classes de forma eficiente.

Por fim, o melhor cenário global observado para o modelo *Convolutional Neural Network* (CNN) foi o D-F (Desbalanceado com fine-tuning), no qual o modelo atingiu acurácia de 0.55, *F1 Weighted* de 0.48 e F1 Macro de 0.22, valores ainda significativamente inferiores aos obtidos pelos demais modelos avaliados neste estudo. Nesse cenário, o *CNN* foi submetido a aplicação de ajuste fino de hiperparâmetros, e as melhoras foram encontradas com os seguintes parâmetros: *(filters = 256, kernel_size = 3, dense_units = 64, dropout_rate = 0.3, optimizer = rmsprop, batch_size = 32 e epochs = 10*).

Esses resultados indicam que, mesmo em sua configuração padrão, o *CNN* não foi capaz de aprender representações discriminativas adequadas para a tarefa de classificação multiclasse de comentários de relatórios de bugs. Tal limitação pode ser atribuída à combinação entre o desbalanceamento das classes, o volume restrito de exemplos por classe e a natureza altamente esparsa e heterogênea dos dados textuais, reforçando uma inadequação de modelos baseados em aprendizado profundo para o contexto específico analisado neste estudo.

7.1.1 Panorama geral dos resultados dos modelos supervisionados

A avaliação integral dos seis modelos de treinamento supervisionados utilizados no experimento de classificação de comentários revelou um conjunto de padrões consistentes sobre o comportamento dos modelos diante do *dataset* de comentários de relatórios de bugs, que é altamente desbalanceado e composto por dez classes de comentários com grande assimetria entre as classes majoritárias e as minoritárias. Entre os modelos analisados, *Random Forest*, *Logistic Regression*, *Support Vector Machine* (SVM) e *Naive Bayes* se destacaram como os modelos de melhor desempenho geral. A Figura 28 apresenta a comparação das acurácias obtidas no melhor cenário de cada

modelo, conforme definido nas subseções anteriores. Observa-se que os modelos *Random Forest*, *Logistic Regression* e *SVM* alcançaram as maiores acurácias nos seus melhores cenários, todas com valor de 0.86, seguidos pelo modelo *Naive Bayes*, que atingiu acurácia de 0.84 em seu melhor cenário.

O modelo *Decision Tree* apresentou desempenho inferior, com acurácia máxima de 0.75, indicando maior sensibilidade ao desbalanceamento das classes. Por sua vez, o modelo *Convolutional Neural Network* (CNN) registrou os piores resultados, com acurácia máxima de 0.54, evidenciando dificuldades significativas em capturar padrões textuais relevantes mesmo nas classes majoritárias. Esses resultados reforçam que, no contexto deste estudo, modelos clássicos de aprendizado supervisionado apresentam maior robustez e adequação para a classificação de comentários de relatórios de bugs quando comparados a abordagens baseadas em aprendizado profundo.

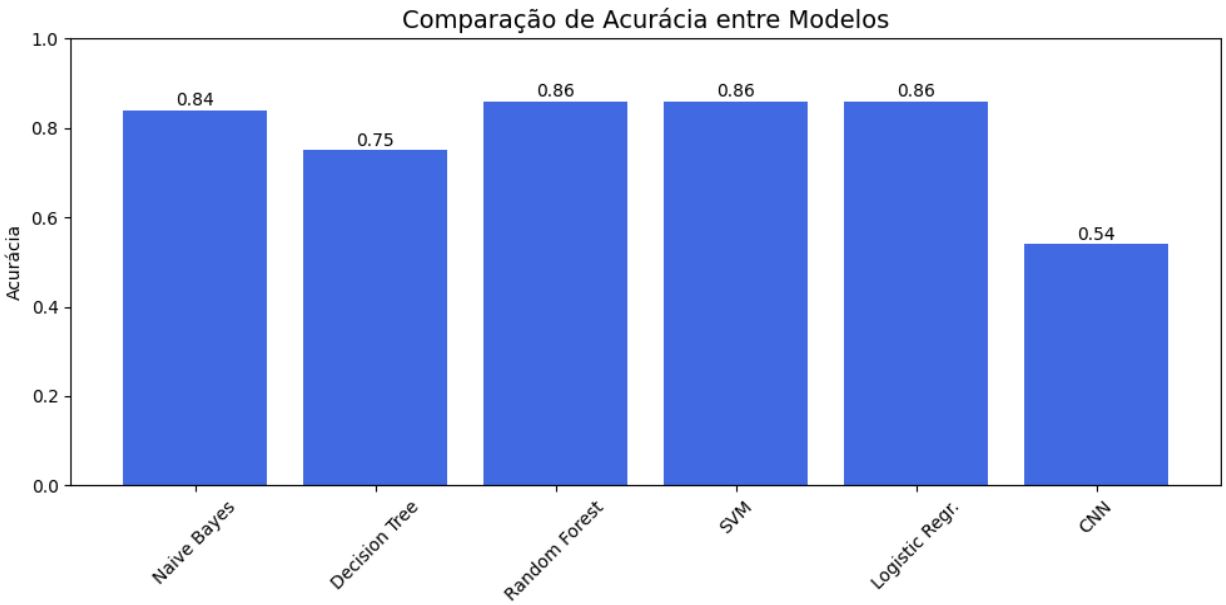


Figura 28: Comparação de Acurácia entre os modelos supervisionados.

A análise da métrica de *F1 Weighted* demonstrou que ela acompanha os altos valores de acurácia dos mesmos modelos, demonstrando que os modelos mantiveram capacidades boas para identificar principalmente as classes majoritárias de comentários de relatórios de bugs. A Figura 29 mostra o *F1 Weighted* como uma métrica global mais sensível à interpretação das classes de comentários. Os modelos *Random Forest*,

Logistic Regression e *SVM* se destacam com valor de *F1 Weighted* em 0.85, seguidos pelo modelo *Naive Bayes* com 0.82. O modelo *Decision Tree* apresenta *F1 Weighted* reduzido com valor de 0.72, enquanto o *CNN* aparece como o pior modelo com um valor de 0.51, demonstrando que mesmo considerando o peso das classes majoritárias, sua capacidade discriminativa permanece baixa para as classes minoritárias.

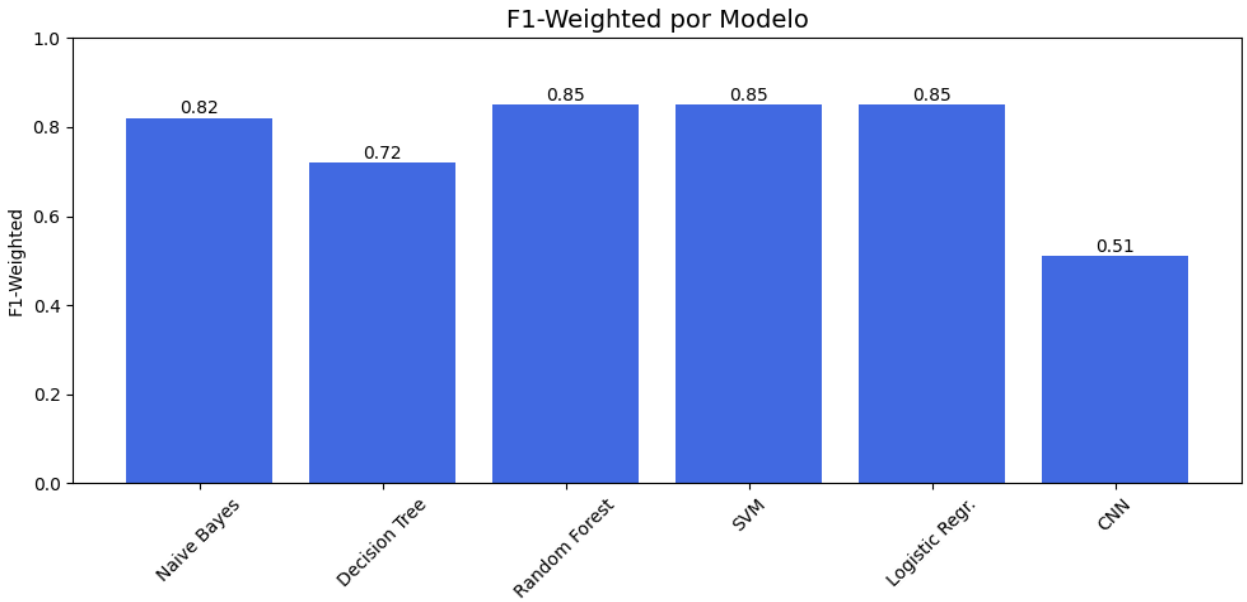


Figura 29: Comparação de F1 Weighted entre os modelos supervisionados

A análise das macro-métricas F1 Macro, Revocação Macro e Precisão Macro evidencia que todos os modelos apresentam queda significativa de desempenho nas classes minoritárias, resultado coerente com a distribuição assimétrica do *dataset*. Em praticamente todos os modelos, a F1 Macro raramente ultrapassa 0.55, mesmo nos melhores cenários, indicando dificuldade em capturar padrões das classes menos representadas. A Figura 30 apresenta uma comparação dos valores de F1 Macro sendo uma métrica essencial para avaliar a performance dos modelos em cenários de classes desbalanceadas, pois calcula o F1 de cada classe individualmente e tira a média não ponderada. Os modelos *Logistic Regression* e *SVM* são os melhores modelos nessa perspectiva, com valores em 0.56 e 0.57 respectivamente, seguidos pelo modelo *Random Forest* com 0.55 e pelo modelo *Naive Bayes* com 0.50. O modelo *Decision Tree* atinge valor reduzido com 0.40, e o *CNN* apresenta novamente o pior desempenho

com 0.23, reforçando que o *CNN* tem limitações na identificação de classes minoritárias e no equilíbrio entre precisão e Revocação.

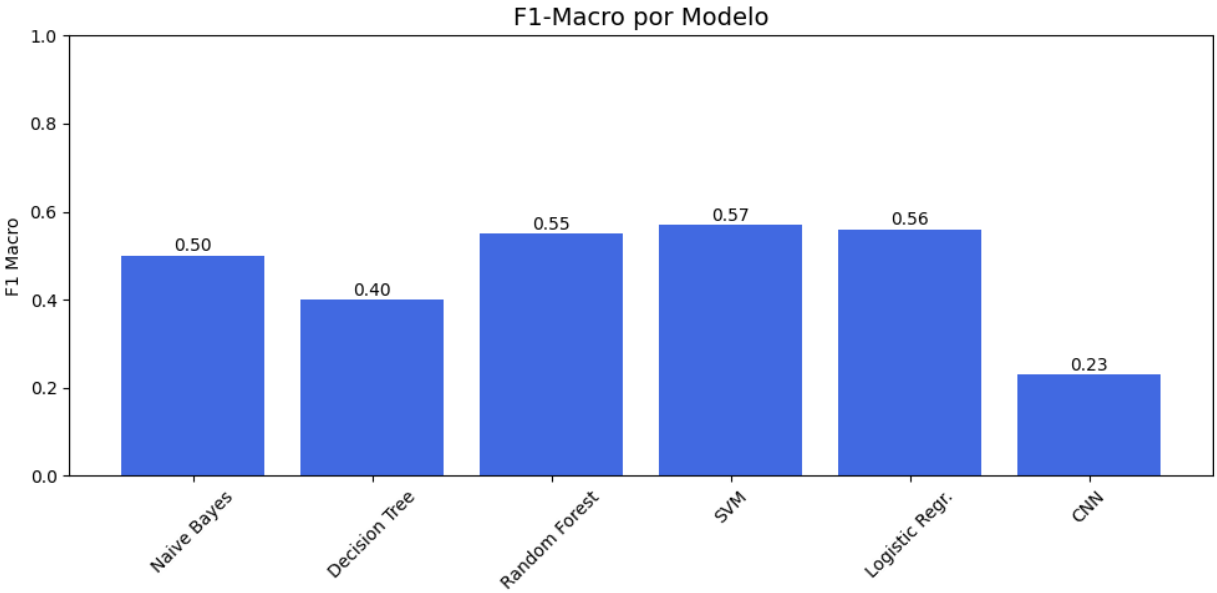


Figura 30: Comparação de F1 Macro entre os modelos supervisionados

A Figura 31 apresenta os valores de Precisão Macro que é a métrica que mede a capacidade do modelo de evitar falsos positivos ao tratar igualmente todas as classes. Os modelos *SVM* e *Random Forest* atingiram os maiores valores com 0.60, seguidos pelos modelos *Naive Bayes* e *Logistic Regression* com valores em 0.58. O modelo *Decision Tree* aparece em um nível inferior com um valor máximo atingindo apenas 0.45, enquanto o *CNN* demonstra baixíssima precisão macro com valor 0.23, indicando grande dificuldade em predizer corretamente classes minoritárias.

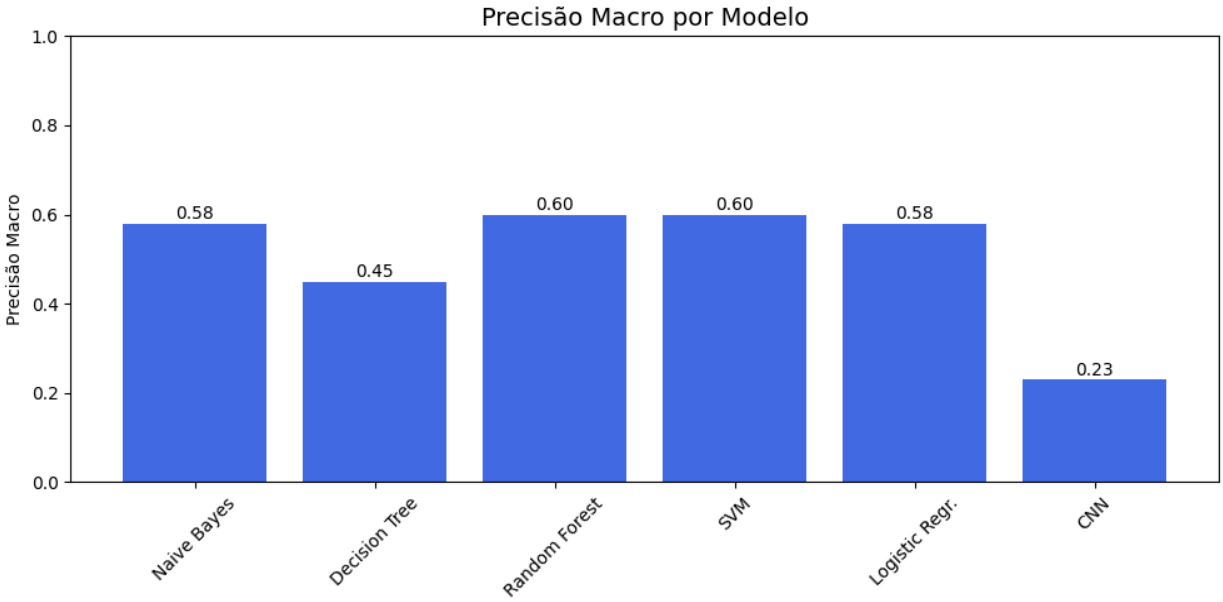


Figura 31: Comparação de Precisão Macro entre os modelos supervisionados.

Por fim, a Figura 32 apresenta os valores de Revocação Macro, essa métrica que consegue capturar a capacidade de identificar corretamente exemplos das classes de comentários, especialmente importante para analisar sensibilidade nas classes minoritárias. Os modelos *Logistic Regression* e *SVM* seguem sendo os melhores modelos com valores de 0.57 e 0.58 respectivamente, seguidos pelo modelo *Random Forest* com valor de 0.53 e *Naive Bayes* com valor de 0.47. O modelo *Decision Tree* apresenta desempenho inferior com 0.41, e o *CNN* permanece como o mais fraco com 0.25, evidenciando sua incapacidade de capturar adequadamente a distribuição do dataset.

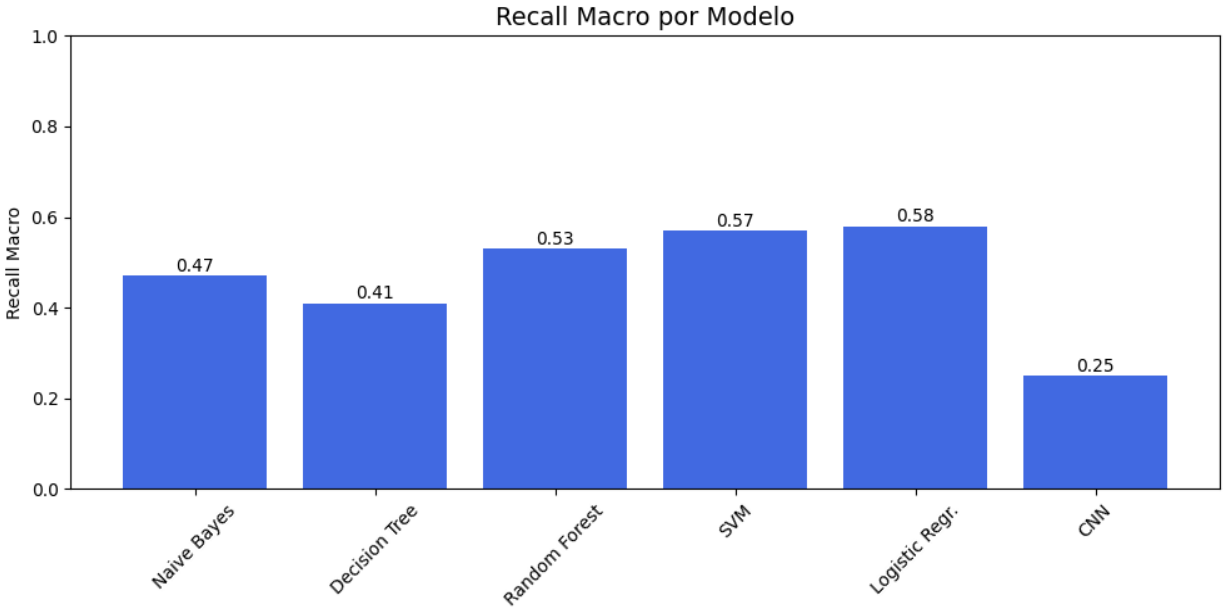


Figura 32: Comparação de Revocação (recall) Macro entre os modelos supervisionados

No contexto desta tarefa, a análise conjunta de *precision* e *recall* é particularmente importante, pois a acurácia e o F1 Weighted permanecem fortemente influenciados pelas classes majoritárias. Isso fica evidente no fato de que modelos como *Random Forest*, *Logistic Regression* e *SVM* atingem acurácia em torno de 0,86 e *F1 Weighted* em torno de 0,85, enquanto o *F1 Macro* permanece limitado a aproximadamente 0,55–0,57, indicando dificuldade persistente nas classes menos frequentes.

Nesse cenário, a *precision* macro mostra o quão confiáveis são as predições entre todas as classes, por exemplo, SVM e Random Forest alcançam os maiores valores, em torno de 0,60, sugerindo menor incidência de falsos positivos ao tratar igualmente classes majoritárias e minoritárias. Já o *recall* macro revela a capacidade de recuperar instâncias reais das classes menos representadas, aspecto crítico porque comentários tecnicamente relevantes, como perguntas, diretrizes e soluções parciais, perdem utilidade prática se forem sistematicamente ignorados pelo classificador. Assim, no presente estudo, *precision* é essencial para avaliar a confiabilidade da tipagem automática, enquanto *recall* é decisivo para medir o quanto o modelo consegue, de fato, capturar as classes raras que mais interessam do ponto de vista analítico e operacional.

Já o modelo *Convolutional Neural Network* (CNN) apresentou desempenho consistentemente inferior, se mostrando como o pior modelo entre os seis avaliados. Suas acurácias variaram entre 0.24 e 0.54, muito abaixo dos demais algoritmos, e os valores de F1 Macro raramente ultrapassam valores acima de 0.28, evidenciando assim a dificuldade tanto para as classes majoritárias quanto para as minoritárias. Mesmo com técnicas de balanceamento, o modelo *CNN* não conseguiu construir representações discriminativas eficazes, o que sugere que seu uso em tarefas com datasets moderadamente pequenos e altamente desbalanceados pode ser limitado.

A leitura de *precision* e *recall* também ajuda a compreender por que o desempenho do CNN foi tão inferior. Mesmo em seu melhor cenário, o modelo apresentou *precision* macro de apenas 0,22 e *recall* macro de 0,25, valores muito abaixo dos observados nos modelos clássicos, evidenciando simultaneamente baixa confiabilidade nas predições e baixa capacidade de recuperar instâncias reais das diferentes classes. Em termos práticos, isso significa que o CNN não apenas erra com frequência, mas também distribui esses erros de forma pouco controlada entre as classes, comprometendo tanto a identificação correta de classes minoritárias quanto a utilidade do modelo para apoiar análises posteriores.

Por fim, a junção de desempenho global, estabilidade em diferentes cenários e equilíbrio entre as métricas de acurácia e *F1 Weighted* indicam que os melhores modelos do experimento foram o modelo *Random Forest*, seguido pelos modelos *Logistic Regression* e *SVM*. Esses modelos demonstraram maior capacidade de adaptação ao desbalanceamento natural dos dados, mantendo acurácia elevada nas classes mais prevalentes sem degradar severamente as classes minoritárias, embora estas ainda permanecem um desafio claro é recorrente em todo o conjunto de experimentos. Em resumo, os resultados mostram que a eficácia dos modelos está fortemente condicionada à capacidade de discriminação das classes majoritárias que direcionam grande parte da acurácia e do *F1 Weighted*. Em contrapartida, as classes minoritárias continuam apresentando baixo desempenho, mesmo com técnicas de balanceamento e ajustes de hiperparâmetros, evidenciando assim a necessidade de estratégias adicionais para tentar melhorar os resultados.

7.1.2 Análise por classes de comentários

Além das análises das métricas de acurácia e *F1 Weighted* das seções anteriores, é fundamental observarmos como cada modelo se comporta a nível de classe de comentários, especialmente por estarmos em um cenário com dez tipos de comentários e forte desbalanceamento entre esses tipos. Neste subcapítulo, aprofundamos a análise comparando o melhor modelo de experimento de classificação de comentários com treinamento supervisionado, o modelo Random Forest no cenário S-F (*SMOTE* com fine-tuning), com o pior modelo do experimento de classificações de comentários, o modelo *CNN* no cenário S-NF (*SMOTE* sem fine-tuning). O propósito deste subcapítulo é entender não só "quem acerta mais", mas quais tipos de comentários são bem representados e quais continuam problemáticos, até mesmo nos melhores cenários.

A inclusão do pior modelo no cenário comparativo não tem apenas caráter ilustrativo, mas diagnóstico. A análise do modelo com desempenho inferior permite identificar padrões de colapso, limitações estruturais da arquitetura adotada e dificuldades específicas associadas a determinadas classes. Essa comparação contribui para evidenciar que os resultados obtidos pelo melhor modelo não são triviais nem decorrentes apenas do desbalanceamento dos dados, mas sim de diferenças efetivas na capacidade de modelagem das abordagens avaliadas.

Random Forest no cenário S-F (SMOTE com fine-tuning): desempenho por classe

A Figura 33 apresenta os valores de *F1-Score* por classe de comentários para o modelo *Random Forest* no cenário S-F (*SMOTE* com fine-tuning). Observa-se um padrão consistente no desempenho do modelo *Random Forest*, diretamente relacionado às características intrínsecas de cada classe de comentários:

- As classes de comentários: B (*Bot Comment*), DE (*Description*) e N (*No Text*) apresentam os maiores valores de F1-Score, o que pode ser explicado pela alta frequência dessas classes no dataset e pela regularidade estrutural dos seus textos. Comentários de bots seguem padrões sintáticos repetitivos, descrições

iniciais de bugs apresentam vocabulário mais rico e consistente, e comentários do tipo *No Text* possuem baixa variabilidade textual, fatores que facilitam a aprendizagem do modelo;

- A classe de comentários U (*Update*) também apresenta desempenho elevado, indicando que o modelo consegue identificar com eficiência comentários associados a mudanças de status ou atualizações do relatório. Isso ocorre porque esse tipo de comentário tende a conter marcadores linguísticos recorrentes, como referências a estados do bug, versões, confirmações de correção ou encerramento, tornando sua identificação mais direta;
- As classes E (*Error*) e G (*Guideline*), apresentam desempenho intermediário. Embora contenham informações técnicas relevantes, essas classes exibem maior variabilidade lexical e semântica, além de sobreposição parcial com outras classes, como *Discussion* e *Partial Solution*, o que dificulta uma separação mais precisa por parte dos modelos;
- As classes claramente minoritárias, PC (*Partial Solution With Code*), POC (*Partial Solution Without Code*) e Q (*Question*) apresentam os menores valores de *F1-Score*. Esse comportamento está diretamente associado ao baixo número de instâncias disponíveis e à ambiguidade semântica desses comentários, que frequentemente compartilham características com classes como *Discussion* e *Guideline*. Além disso, soluções parciais e perguntas tendem a ser expressas de forma menos padronizada, dificultando a aprendizagem de padrões discriminativos estáveis, mesmo em cenários com balanceamento via *SMOTE*.

Esses resultados mostram que o modelo Random Forest não colapsa completamente nas classes minoritárias mesmo que o desempenho nelas seja inferior ao das majoritárias, o modelo consegue ao menos distinguir alguns padrões e manter F1 acima de zero para todas as categorias.

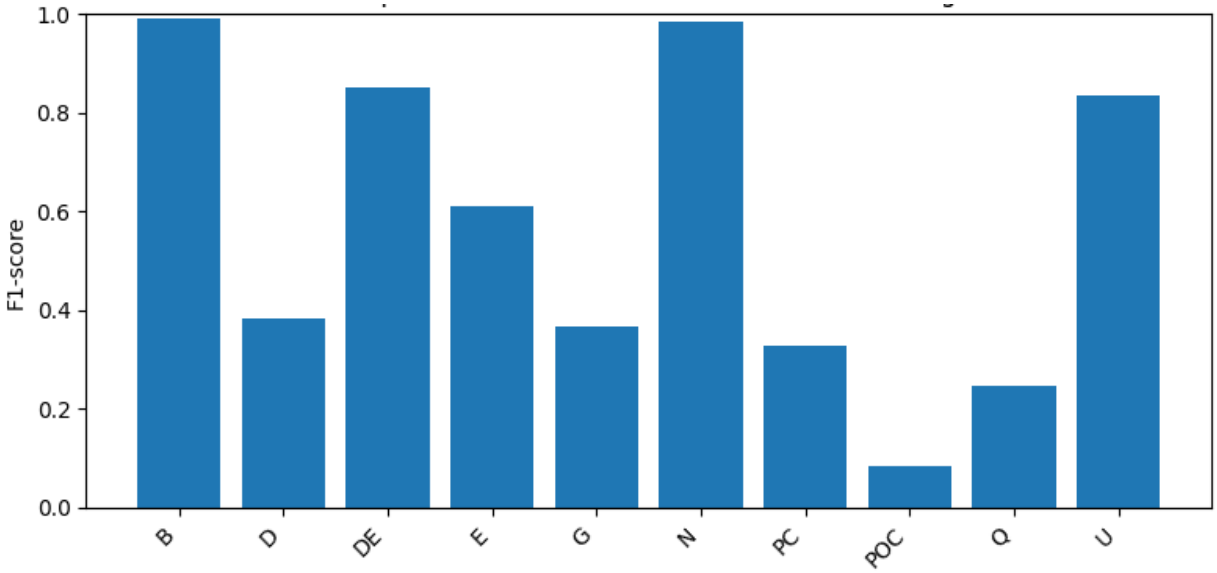


Figura 33: F1 por classe de comentário no modelo SMOTE, cenário S-F.

A Figura 34 aprofunda a visão ao mostrar as principais confusões do modelo *Random Forest* de forma normalizada. O eixo horizontal representa a proporção relativa de erros, indicando a fração de ocorrências em que uma classe verdadeira foi incorretamente predita como outra classe específica. Por exemplo, um valor de 0.1 indica que aproximadamente 10% das instâncias daquela classe foram confundidas com a classe correspondente, enquanto valores próximos de 0.4 representam confusões mais frequentes, envolvendo cerca de 40% dos erros daquela classe.

Várias dessas confusões são semanticamente plausíveis, reforçando que parte dos erros do modelo deriva da proximidade conceitual entre as classes:

- Comentários do tipo D (*Discussion*) são frequentemente confundidos com Q (*Question*) e G (*Guideline*). Esse comportamento faz sentido, pois na prática muitas discussões incluem perguntas e sugestões misturadas;
- A classe DE (*Description*) aparece confundida com PC (*Partial Solution With Code*) e POC (*Partial Solution Without Code*), indicando que, quando há descrição de erro junto com uma possível solução parcial, o modelo às vezes prioriza o aspecto do "erro", às vezes do aspecto "solução";

- Há confusões recorrentes entre G (*Guideline*) e POC (*Partial Solution Without Code*), o que também é coerente, já que *guidelines* e soluções parciais sem código tem formatos textuais similares.

Embora essas confusões indiquem limitações do modelo, elas são, em grande parte, erros semanticamente compreensíveis, uma vez que ocorrem predominantemente entre classes conceitualmente próximas. Isso sugere que o modelo, mesmo ao errar, tende a fazê-lo dentro de um mesmo campo semântico, reforçando a coerência interna da taxonomia proposta e a plausibilidade dos padrões aprendidos.

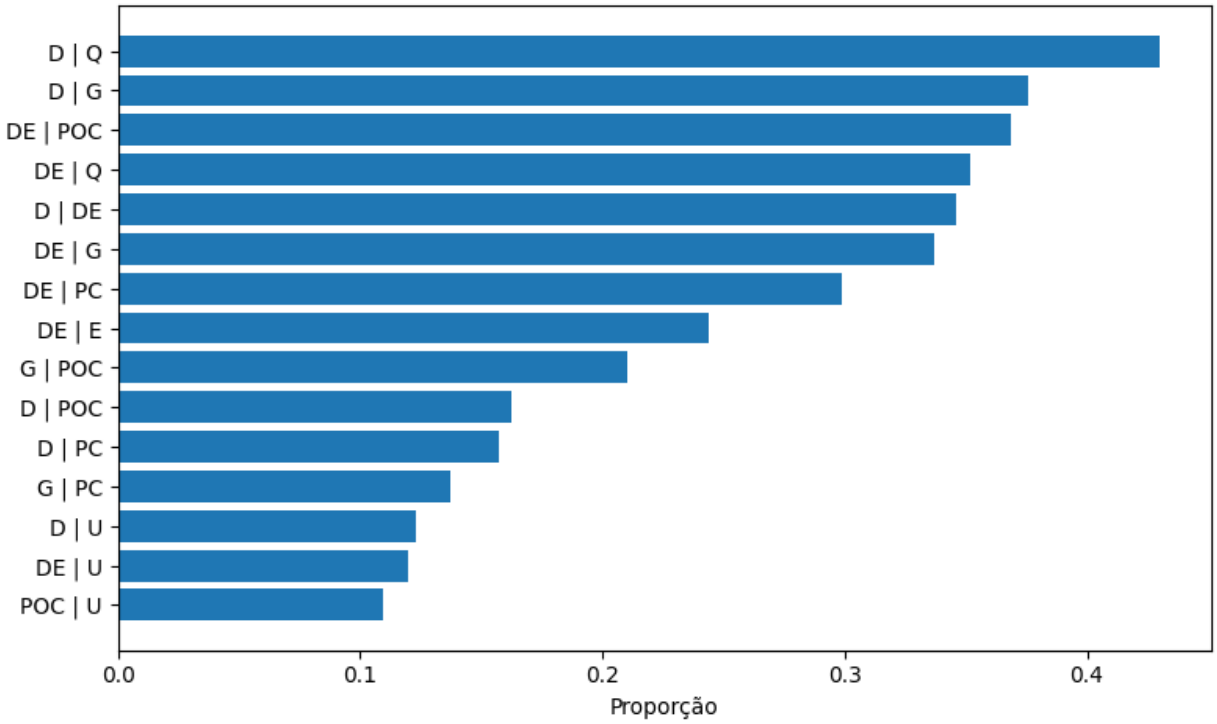


Figura 34: Top 15 confusões de classes de comentários do modelo Random Forest cenário S-F.

Por fim, a Figura 35 apresenta a matriz de confusão do Random Forest no cenário S-F (SMOTE com fine-tuning). Ela reforça o que as métricas já apontavam:

- Nas classes de comentários B (*Bot Comment*), DE (*Description*), N (*No Text*) e U (*Update*), a maior parte da massa de probabilidade está concentrada na diagonal, com valores próximos de 0.99, 0.92, 0.98 e 0.76 respectivamente, evidenciando um alto índice de acertos nessas categorias;
- As classes minoritárias apresentam linhas mais espalhadas, com parte significativa dos exemplos sendo absorvida pelas classes majoritárias. Ainda assim, as classes de comentários PC (*Partial Solution With Code*), POC (*Partial Solution Without Code*) e Q (*Question*) apresentam parcelas não desprezíveis na diagonal (por exemplo, PC com acerto em torno de 0.30), o que significa que o modelo consegue, em pelo menos uma fração dos casos, reconhecer explicitamente que se trata de uma solução parcial ou de uma pergunta, e não apenas classificá-las como discussão genérica.

Essa matriz de confusão mostra que o Random Forest consegue equilibrar a modelagem das classes majoritárias com algum nível de sensibilidade às classes menos representadas, ainda que estas continuem sendo o ponto fraco do sistema.

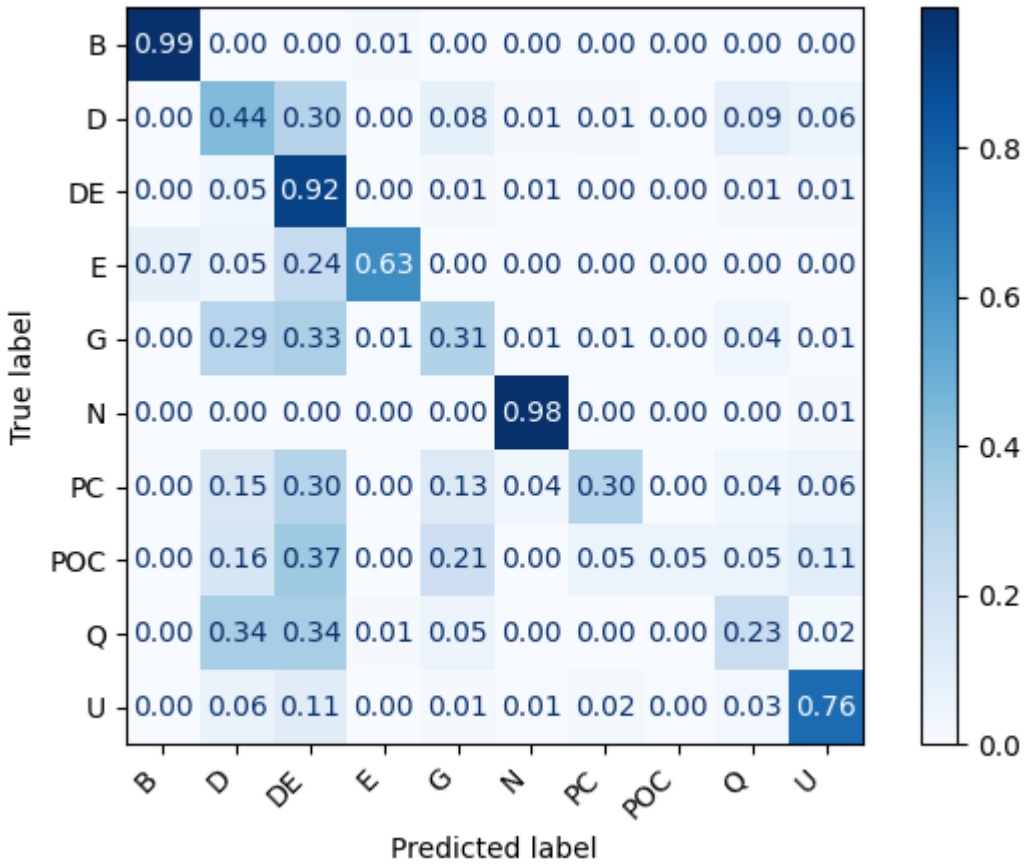


Figura 35: Matriz de confusão das classes de comentários no modelo Random Forest no cenário S-F.

Mesmo no melhor cenário experimental, utilizando *Random Forest* com aplicação de *SMOTE* para balanceamento das classes, foram observados casos pontuais de classificação incorreta envolvendo comentários semanticamente ambíguos. A Figura 36 apresenta um exemplo desse tipo de ocorrência. O comentário em questão foi rotulado manualmente como *Discussion*, porém foi classificado pelos modelos supervisionados, incluindo o Random Forest, como *Description*.

O conteúdo do comentário analisa conceitualmente o fluxo de estados de um sistema, discute transições possíveis e sugere aprofundamento na documentação do comportamento da aplicação, sem apresentar código explícito ou relatar erro específico. Embora a intenção comunicativa envolve debate técnico, característica associada à

classe *Discussion*, a estrutura textual assume formato predominantemente explicativo e descritivo, o que pode ter levado os modelos a identificá-lo como *Description*.

Esse caso evidencia que, mesmo em cenários com desempenho global elevado, a distinção entre classes semanticamente próximas permanece um desafio intrínseco à tarefa. A fronteira entre descrição técnica e discussão argumentativa nem sempre é claramente delimitada, especialmente em contextos colaborativos como relatórios de bugs. Assim, parte das classificações incorretas observadas nas matrizes de confusão decorre não apenas de limitações algorítmicas, mas também da própria sobreposição conceitual entre categorias definidas na taxonomia.

Em resposta à QP1, os resultados obtidos indicam que a classificação automática dos comentários é viável quando utilizados modelos supervisionados tradicionais aliados a técnicas de balanceamento, como o SMOTE. O cenário com Random Forest apresentou o melhor equilíbrio entre as classes, demonstrando capacidade consistente de capturar padrões linguísticos associados às diferentes funções comunicativas definidas na taxonomia. Embora persistam confusões entre classes semanticamente próximas, os resultados confirmam que os comentários podem ser classificados de forma sistemática e automatizada com desempenho satisfatório.


Chris Karlof [ckarlof]
Comment 20 • 11 years ago

Mark, I'm of the opinion that (currently) once the remote page submits the "login" message to the browser, it's done -- at that point it's dead code. I think to properly think about this we need consider all the states and what the transitions should be.

For the simplest case (creating an account and the user verifies her email), jgruen has indicated that about:accounts should transition to the "Manage" screen, which doesn't involve any messaging to the iframe (see pg 6 of https://www.dropbox.com/s/1dm7d03jk8rowc/desktop%20touch%20up.pdf).

I'm going to try to document the state machine more.

Figura 36: Exemplo de comentário classificado pelo modelo Random Forest.

Convolutional Neural Network (CNN) no cenário S-NF (*SMOTE* sem fine-tuning): falhas sistemáticas por classe de comentários

A Figura 37 apresenta o F1-Score por classe para o modelo *CNN* no cenário S-NF (SMOTE sem fine-tuning). O comportamento é quase o oposto do observado no Random Forest:

- Apenas duas classes de comentários se destacam: N (*No Text*), com F1 próximo de 0.94, e U (*Update*), com F1 em torno de 0.58;
- A classe de comentário B (*Bot Comment*) atinge um F1 moderado de valor 0.45;
- Para todas as demais classes D (*Discussion*), DE (*Description*), E (*Error*), G (*Guideline*), PC (*Partial Solution With Code*), POC (*Partial Solution Without Code*) e Q (*Question*), o F1 é praticamente nulo com valores em torno de 0.00 a 0.05.

Na prática, esses dados mostram que o modelo *CNN* não conseguiu aprender padrões úteis para quase todas as categorias que exigem interpretação textual mais rica, e só "sobrevive" nas classes com estrutura muito recorrente e pouco variada, como mensagens de bots e comentários vazios.

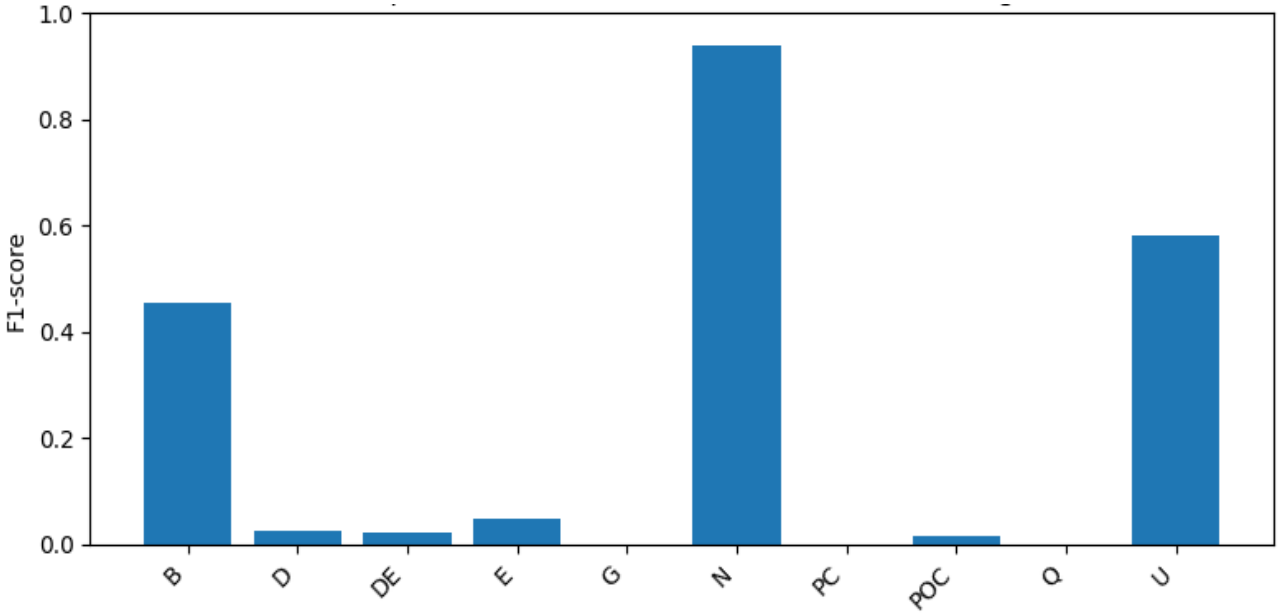


Figura 37: F1 por classe de comentário no modelo CNN, cenário S-NF.

A Figura 38 mostra as principais confusões do modelo *CNN*, explicitando esses colapsos. Observamos que:

- A classe de comentários B (*Bot Comment*) é massivamente confundida com E (*Error*), PC (*Partial Solution With Code*), Q (*Question*) e POC (*Partial Solution Without Code*), indicando que textos de bot são frequentemente interpretados como descrições de erro ou soluções parciais, o que não faz sentido do ponto de vista semântico, porém em análises manuais, os comentários feitos por bots destacam casos de falha em testes automáticos que são executados, embasando a confusão do modelo em confundir mensagens de bots com erros;
- Comentários originalmente classificados como E (*Error*) ou DE (*Description*) acabam frequentemente classificados como outras categorias, inclusive B (*Bot Comment*) ou N (*No Text*), o que mostra que o modelo não conseguiu separar sequer os papéis mais centrais no fluxo do relatório de bug;
- Há confusões intensas envolvendo a classe E (*Error*) com múltiplas classes, o que indica que o modelo "não sabe o que fazer" com descrições de erro: ora os enxerga como discussão, ora como atualização, ora como outras coisas.

Diferentemente do Random Forest, aqui as confusões não são apenas "semânticas" ou "discutíveis"; elas revelam uma ausência clara de fronteiras entre as classes.

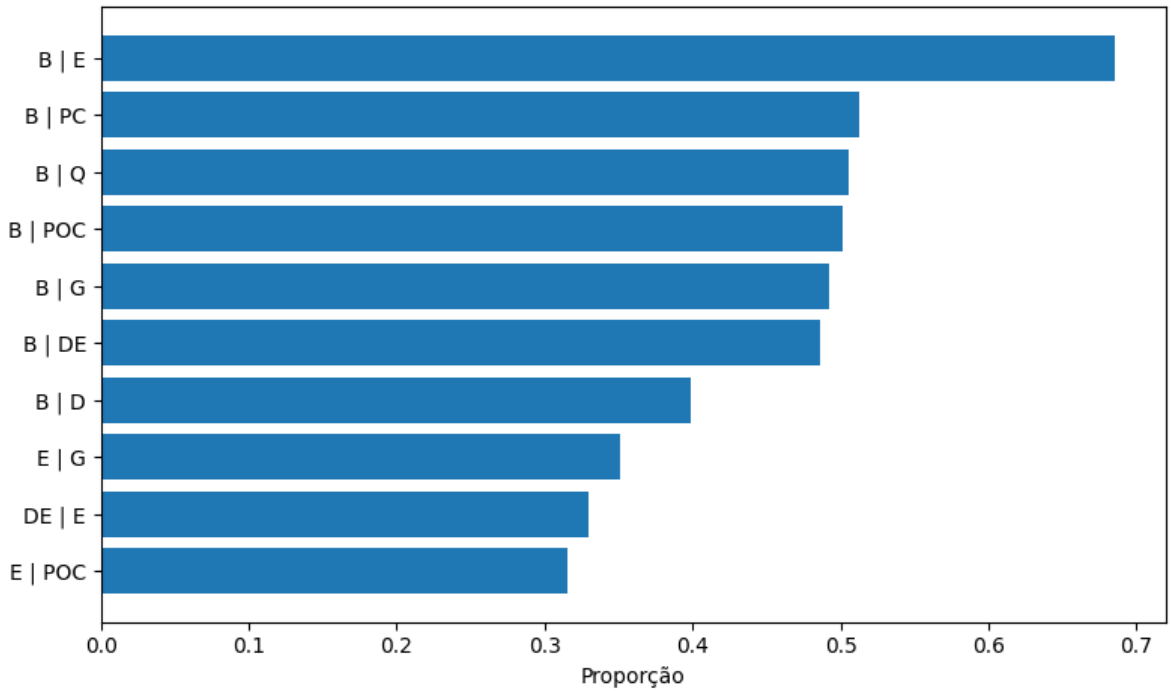


Figura 38: Top 10 confusões de classes de comentários do modelo CNN no cenário S-NF.

Por fim, a Figura 39 mostra a matriz de confusão do modelo *CNN* no cenário D-NF. Visualmente, é possível notar:

- Grande parte das linhas concentra-se em poucas colunas, principalmente das classes N (*No Text*) e U (*Update*), o que sugere um colapso de decisão: o modelo tende a prever sistematicamente essas classes, independentemente da classe verdadeira;
- Mesmo em classes majoritárias como B (*Bot Comment*) e DE (*Description*), a massa na diagonal é modesta, com valores de 0.55 e 0.33 respectivamente, com grande parte dos exemplos sendo desviados para outras categorias;
- As classes de comentários minoritárias PC (*Partial Solution With Code*), POC (*Partial Solution Without Code*), Q (*Question*), E (*Error*) e G (*Guideline*)

praticamente não aparecem com valor relevante na diagonal, o que significa que o modelo *CNN* quase nunca consegue acertar essas classes.

A matriz de confusão confirma que, com a arquitetura e o esquema de vetorização utilizados, o modelo *CNN* não conseguiu construir representações discriminativas adequadas. Em vez de aprender padrões semânticos robustos, ela se apoia em alguns sinais superficiais que a empurram para as classes mais frequentes ou mais "fáceis" de reconhecer, como a classe N (*No Text*).

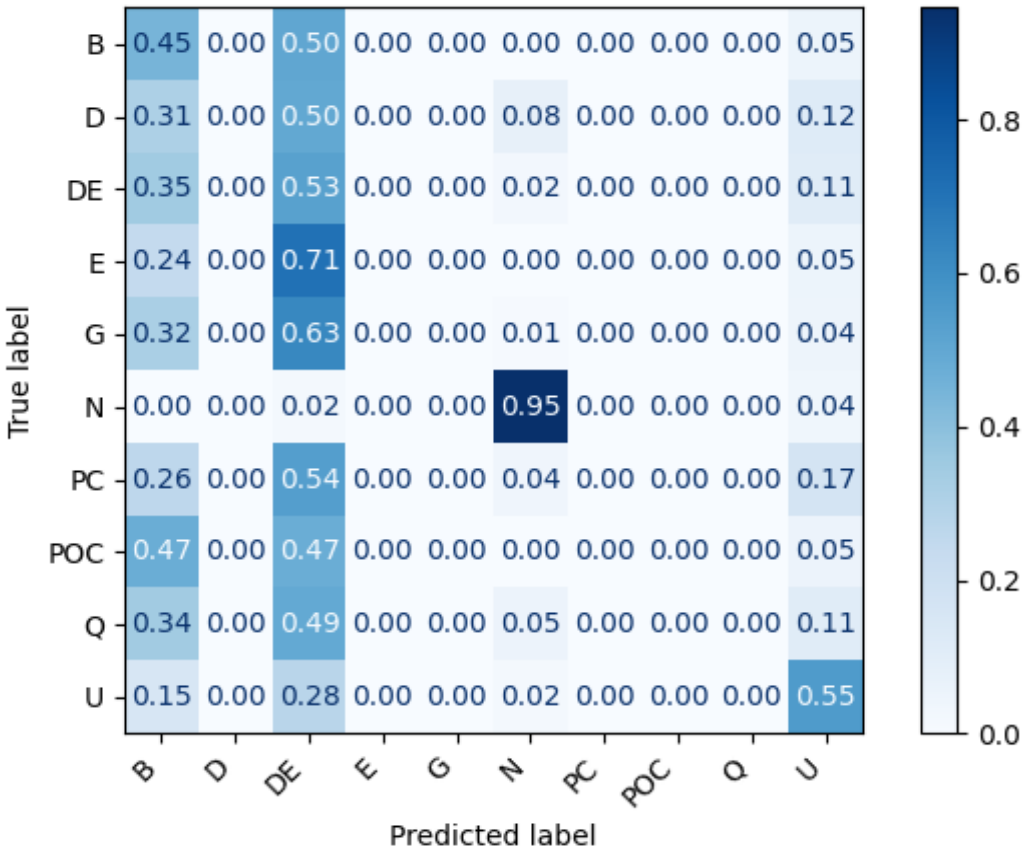


Figura 39: Matriz de confusão das classes de comentários no modelo CNN no cenário D-NF.

Resumo da análise por classes de comentários

Com base nos resultados coletados e analisados nas seções anteriores, tem-se que:

- O modelo Random Forest consegue manter altos F1-Score nas classes majoritárias e ainda apresenta alguma capacidade de captar classes minoritárias, mesmo que com desempenho mais modesto. Suas confusões são em grande parte coerentes com a proximidade semântica entre os tipos de comentário;
- Os modelos Naive Bayes, Regression Logistic, SVM e Decision tree acompanham o modelo Random Forest em resultados próximos com os mesmos teores de confusões entre classes de comentários majoritárias e minoritárias;
- O modelo *CNN*, por outro lado, falha de forma sistemática: colapsa em poucas classes, têm F1 quase nulo nas classes minoritárias e produz um padrão de confusões que indica ausência de separação estrutural entre os tipos de comentário.

Essa análise por classe reforça o resultado global do experimento com modelos de treinamento supervisionados: em um cenário com texto curto, dataset moderado e forte desbalanceamento, modelos clássicos como *Random Forest* são muito mais adequados do que o modelo *CNN* simples aplicado diretamente sobre vetores TF-IDF.

7.2 Modelos Não Supervisionados

Os modelos não supervisionados utilizados neste estudo foram o *DBSCAN*, *HDBSCAN* e *K-Means* possuem pressupostos e comportamentos distintos quanto à formação de agrupamentos, número de clusters e tratamento de amostras atípicas. De modo geral, esses algoritmos buscam identificar estruturas latentes nos dados sem utilizar rótulos previamente definidos, o que torna sua aplicação particularmente desafiadora em conjuntos de dados textuais de alta dimensionalidade e com forte sobreposição semântica, como os comentários de relatórios de bugs.

Para os modelos não supervisionados, cada cenário experimental é definido pela combinação entre a estratégia de distribuição do dataset, incluindo dataset

desbalanceado, *Oversampling*, *Undersampling*, *SMOTE* e *Cluster Centroids*; e a presença ou não de ajuste de hiperparâmetros por meio de busca em grade (*Grid Search*). As siglas utilizadas para identificação dos cenários seguem o padrão: D (Desbalanceado), O (Oversampling), U (Undersampling), S (SMOTE), CC (Cluster Centroids), NF (sem ajuste de hiperparâmetros) e F (com ajuste de hiperparâmetros).

Diferentemente dos modelos supervisionados, os modelos de aprendizado não supervisionado não recebem como entrada os rótulos reais das classes de comentários, buscando identificar estruturas latentes nos dados a partir de critérios internos, como densidade, proximidade e conectividade no espaço vetorial. Como consequência, o número de clusters gerados não é fixo, podendo variar entre cenários e modelos, e não necessariamente coincide com o número de classes da taxonomia proposta.

Em particular, algoritmos baseados em densidade, como o *DBSCAN* e o *HDBSCAN*, identificam regiões densas no espaço vetorial e atribuem amostras que não pertencem a nenhuma dessas regiões ao rótulo *NOISE* (−1). Esse comportamento é esperado em dados textuais com fronteiras semânticas pouco definidas e alta sobreposição entre categorias, como é o caso dos comentários de relatórios de bugs. Já o *K-Means*, por sua vez, exige a definição prévia do número de clusters e atribui todas as amostras a algum agrupamento, não possuindo um mecanismo nativo para identificação de ruído.

Para possibilitar a comparação entre os agrupamentos produzidos pelos modelos não supervisionados e as classes reais obtidas na rotulação manual, foi utilizada a métrica *Adjusted Rand Index* (ARI). O ARI mede o grau de concordância entre dois particionamentos, independentemente do número de clusters, sendo particularmente adequado para cenários em que o modelo gera um número variável de agrupamentos. Seus valores variam no intervalo [−1,1], onde valores próximos de 1 indicam alta correspondência entre clusters e classes reais, valores próximos de 0 indicam concordância aleatória e valores negativos indicam discordância significativa.

Adicionalmente, também foi utilizada a métrica *Silhouette Score*, que avalia a coesão interna e a separação entre os clusters formados, também variando no intervalo [−1,1]. Valores elevados indicam clusters compactos e bem separados, enquanto valores próximos de zero ou negativos sugerem sobreposição ou estrutura fraca de

agrupamento. O uso conjunto dessas métricas permite analisar tanto a qualidade estrutural dos clusters quanto sua aderência às classes reais de comentários, fornecendo uma visão complementar do desempenho dos modelos não supervisionados avaliados.

DBSCAN

O modelo DBSCAN foi avaliado em todos os cenários definidos para os modelos não supervisionados, considerando diferentes estratégias de distribuição do *dataset* e a presença ou não de ajuste de hiperparâmetros. Por se tratar de um algoritmo baseado em densidade, o *DBSCAN* não requer a definição prévia do número de clusters e permite a identificação explícita de amostras classificadas como ruído (*NOISE*), característica relevante para dados textuais de alta dimensionalidade e forte sobreposição semântica, como os comentários de relatórios de bugs. A Tabela 10 apresenta os valores obtidos para as métricas *Adjusted Rand Index* (ARI) e *Silhouette Score* em cada cenário avaliado. De forma geral, os resultados indicam que o *DBSCAN* apresentou dificuldades em capturar adequadamente a estrutura semântica subjacente do *dataset*, refletidas pelos baixos valores de ARI e pelos valores de *Silhouette* frequentemente próximos de zero ou negativos.

Tabela 10: Métricas dos cenários do modelo DBSCAN.

Cenário	*Silhouette*	ARI
D-NF	0.27	0.37
D-F	0.15	0.29
O-NF	0.64	0.31
O-F	0.67	0.27
CC-NF	-0.10	0.01
CC-F	0.24	0.29
U-NF	0.11	0.06
U-F	0.23	0.29
S-NF	-0.11	0.06
S-F	0.007	0.08

Ao analisar as métricas em conjunto, observa-se um comportamento distinto entre qualidade estrutural dos clusters e aderência às classes reais. Os cenários O-NF e O-F apresentaram os melhores resultados em termos de coesão e separação interna (*Silhouette Score* de 0.64 e 0.67, respectivamente), indicando clusters mais bem definidos no espaço vetorial. Por outro lado, a maior aderência às classes reais, medida pelo ARI, ocorreu no cenário D-NF (ARI = 0.37; Silhouette Score= 0.27), sugerindo que a distribuição original preserva melhor a correspondência com a taxonomia manual, ainda que com estrutura de agrupamento menos coesa do que nos cenários com *Oversampling*.

Nos cenários desbalanceados (D-NF e D-F), o *DBSCAN* apresentou um comportamento mais alinhado à distribuição original dos dados. Em particular, o cenário D-NF obteve o maior valor de ARI (0.37) entre todos os cenários avaliados, indicando que, mesmo sem intervenções artificiais na distribuição das classes, o modelo conseguiu capturar parte da correspondência entre os agrupamentos formados e as classes reais da taxonomia. Esse resultado sugere que a preservação da densidade

natural dos dados favorece a identificação de regiões semanticamente consistentes, ainda que os valores de *Silhouette Score* permaneçam moderados (0.27), refletindo uma estrutura de clusters menos compacta quando comparada a cenários com redistribuição artificial.

Os cenários com *Oversampling* (O-NF e O-F) apresentaram os maiores valores de *Silhouette Score*, atingindo 0.64 e 0.67, respectivamente. Esses resultados indicam maior coesão interna e melhor separação entre os clusters formados. No entanto, os valores de ARI nesses cenários permaneceram moderados, variando entre 0.27 e 0.31, evidenciando que, apesar de estruturalmente mais bem definidos, os clusters não correspondem de forma satisfatória às classes reais de comentários.

Nos cenários com *Undersampling* (U-NF e U-F), o desempenho do *DBSCAN* foi significativamente inferior. O cenário U-NF apresentou ARI de apenas 0.06 e *Silhouette Score* de 0.11, enquanto o ajuste de hiperparâmetros no cenário U-F elevou esses valores para 0.29 e 0.23, respectivamente. Ainda assim, os resultados permanecem abaixo dos observados nos cenários desbalanceados e com *Oversampling*, indicando que a remoção de grandes quantidades de amostras compromete a formação de regiões densas no espaço vetorial.

Os cenários envolvendo *SMOTE* (S-NF e S-F) também não apresentaram resultados satisfatórios. Os valores de ARI permaneceram muito baixos, variando entre 0.06 e 0.08, com *Silhouette Score* negativo ou próximo de zero. Esse comportamento sugere que a geração de amostras sintéticas distorce a estrutura de densidade necessária para o funcionamento adequado do *DBSCAN*.

Por fim, os cenários com *Cluster Centroids* (CC-NF e CC-F) apresentaram os piores desempenhos globais. No cenário CC-NF, o ARI atingiu apenas 0.01, com *Silhouette Score* negativo (−0.10), indicando uma destruição quase completa da estrutura de densidade dos dados. Embora o cenário CC−F tenha apresentado uma melhora relativa (ARI de 0.29 e *Silhouette Score* de 0.24), os resultados ainda não superaram de forma consistente os cenários desbalanceados ou com *Oversampling*.

De maneira geral, os resultados demonstram que o *DBSCAN* apresentou limitações significativas para capturar a estrutura semântica do *dataset* de comentários, sendo particularmente sensível a intervenções na distribuição dos dados. Mesmo nos cenários em que os clusters apresentaram boa coesão interna, a correspondência com as classes reais permaneceu limitada, reforçando os desafios do uso de algoritmos baseados em densidade em dados textuais altamente sobrepostos.

HDBSCAN

A aplicação do modelo *HDBSCAN* apresentou desempenho limitado na tarefa de agrupamento dos comentários com valores moderados de *Adjusted Rand Index* (ARI) e *Silhouette Score* em todos os cenários testados. O método demonstrou sensibilidade tanto ao balanceamento do *dataset* quanto à aplicação de fine-tuning, com variações entre cenários que não foram suficientes para produzir agrupamentos próximos das classes verdadeiras. A Tabela 11 mostra os valores de todas as métricas coletadas em cada cenário executado.

Tabela 11: Métricas dos cenários do modelo HDBSCAN

Cenário	*Silhouette*	ARI
D-NF	0.25	0.44
D-F	0.29	0.37
O-NF	N	N
O-F	N	N
CC-NF	0.07	0.001
CC-F	-0.08	0.03
U-NF	0.09	0.27
U-F	0.10	0.07
S-NF	N	N
S-F	N	N

O melhor cenário observado foi o D-NF (Desbalanceado sem fine-tuning), com valores de ARI de 0.44 e *Silhouette Score* de 0.25 indicando que o modelo se beneficiou da estrutura original dos dados, porém sem alcançar níveis altos de separação ou coerência semântica. No cenário D-F (Desbalanceado com fine-tuning), o HDBSCAN manteve desempenho baixo, com um leve aumento no *Silhouette Score* para 0.29 e queda no ARI para 0.37. Esses resultados sugerem que os ajustes de hiperparâmetros favoreceram uma maior compactação dos clusters internos, mas não necessariamente aproximaram esses agrupamentos das classes reais. Ainda assim, esses valores permanecem insuficientes para caracterizar uma estrutura de clusters bem formada.

Os cenários com *Undersampling* (U-NF e U-F) apresentaram reduções significativas no desempenho. No cenário U-NF (*Undersampling* sem fine-tuning), o ARI ficou com valores de 0.27 e com *Silhouette Score* de 0.09, valores indicativos de

dispersão entre os pontos e dificuldade na definição de grupos estáveis. No cenário com fine-tuning (U-F), o ARI caiu drasticamente para 0.07, mesmo apresentando um valor de *Silhouette Score* levemente maior 0.10, evidenciando que a redução do *dataset* comprometeu severamente a capacidade do modelo de identificar regiões densas no espaço vetorial.

Os cenários com *Cluster Centroids* (CC-NF e CC-F) tiveram os piores resultados gerais. A compressão dos dados em representações sintetizadas reduziu drasticamente a diversidade e a densidade local necessárias para o agrupamento do *HDBSCAN*. No cenário CC-NF (*Cluster Centroids* sem fine-tuning), o ARI foi praticamente nulo 0.001, com *Silhouette Score* de 0.07, enquanto no cenário CC-F (*Cluster Centroids* com fine-tuning), o ARI subiu para 0.03, acompanhado de um *Silhouette Score* negativo -0.08, indicando clusters instáveis e mal definidos.

Destacamos uma limitação relevante nesta parte do experimento com o modelo *HDBSCAN*: os cenários com *Oversampling* e *SMOTE* não puderam ser executados corretamente mesmo em ambientes de alto desempenho. A combinação do aumento significativo da quantidade de *embeddings* com o custo computacional do *HDBSCAN* tornou a execução inviável, mesmo após sucessivas tentativas e ajustes. Essa restrição limita parte da análise comparativa entre estratégias de balanceamento, além de evidenciar desafios práticos no uso desse modelo em datasets expandidos artificialmente.

Por fim, os resultados indicam que o modelo *HDBSCAN* não conseguiu modelar adequadamente a estrutura semântica dos comentários, apresentando dificuldades consistentes na formação de clusters coerentes e próximos à distribuição real das classes.

K-Means

A aplicação do modelo *K-Means* apresentou variação limitada entre os cenários avaliados, com valores de ARI e *Silhouette Score* permanecendo em faixas semelhantes

independentemente da técnica de balanceamento aplicada. Esse comportamento indica que o modelo responde de forma estável às diferentes distribuições do dataset, sem ganhos ou perdas expressivas ao alternar entre cenários desbalanceados ou balanceados. Ainda assim, os resultados se mostraram limitados para a tarefa de agrupamento dos comentários, com valores moderados de *Adjusted Rand Index* (ARI) e *Silhouette Score* em todos os cenários testados. A Tabela 12 mostra os valores de todas as métricas coletadas em cada cenário executado.

Tabela 12: Métricas dos cenários do modelo K-Means.

Cenário	*Silhouette*	ARI
D-NF	0.34	0.47
D-F	0.34	0.47
O-NF	0.18	0.22
O-F	0.18	0.23
CC-NF	0.06	0.13
CC-F	0.06	0.15
U-NF	0.18	0.24
U-F	0.19	0.22
S-NF	0.23	0.17
S-F	0.23	0.17

O melhor resultado ocorreu no cenário D-NF (Desbalanceado com fine-tuning), no qual o modelo alcançou valores de ARI de 0.47 e *Silhouette Score* de 0.34, indicando que a estrutura natural dos *embeddings* favoreceu a formação de clusters mais compactos e com maior aderência às classes reais. Essa configuração destaca que o *K-Means* funciona melhor quando não há manipulação da distribuição original dos dados.

Nos cenários com *Undersampling* (U-NF e U-F), houve queda perceptível no desempenho. O ARI caiu para 0.24 e o *Silhouette Score* para 0.18, mostrando que a remoção de amostras prejudica a definição das fronteiras entre clusters, afetando a qualidade das regiões formadas. Mesmo com a seleção dos melhores hiperparâmetros, os agrupamentos não recuperam as relações presentes no dataset.

Os cenários com *Cluster Centroids* (CC-NF e CC-F) apresentaram desempenho ainda mais fraco que os demais. A compressão dos dados em poucos centróides reduziu a variabilidade interna e produziu clusters artificiais com baixa capacidade de representação. O ARI ficou em 0.13 e o *Silhouette Scor*e em 0.06, indicando ausência de coesão e separabilidade.

Os cenários com *Oversampling* (O-NF e O-F) mostraram uma melhora moderada em relação aos cenários com *Undersampling* e *Cluster Centroids*. O ARI ficou em 0.22 e o *Silhouette Score* em 0.18, valores similares ao cenário *Undersampling* com fine-tuning (U-F), mas ainda distantes do desempenho no dataset original. Embora o *Oversampling* aumente a densidade em regiões marginais do espaço vetorial, ele também introduz redundância que pode favorecer o colapso de clusters, limitando o ganho de desempenho.

Os cenários com *SMOTE* (S-NF e S-F) apresentaram características intermediárias: ARI com valores de 0.17 e *Silhouette Score* de 0.23. Diferente dos cenários com *Oversampling*, o *SMOTE* aumentou a coesão interna dos clusters, refletida em valores de *Silhouette Score* mais altos, mas ao custo de reduzir a aderência às classes verdadeiras, com valores de ARI menores. A interpolação sintética entre comentários tende a aproximar artificialmente pontos sem relação semântica, prejudicando a identificação de fronteiras naturais entre as classes.

7.2.2 Resultados Gerais dos modelos não supervisionados

A análise dos três modelos não supervisionados, *DBSCAN*, *HDBSCAN* e *K-Means*, aplicados ao *dataset* de comentários, revelou um panorama consistente sobre os

desafios de agrupar textos de alta dimensionalidade, esparsos e fortemente desbalanceados. Os resultados mostram que embora cada algoritmo tenha pontos fortes específicos, todos apresentaram limitações marcantes para capturar a estrutura real das classes presentes no *dataset*.

A Figura 40 apresenta a matriz de confusão do melhor cenário do modelo *DBSCAN*, identificado como o cenário O-F (*Oversampling* com fine-tuning). Esse cenário apresentou os melhores resultados em termos de coesão e separação interna dos clusters, conforme indicado pelos maiores valores de *Silhouette Score* entre os cenários avaliados, ainda que a aderência às classes reais permaneça limitada. A Figura 40 destaca o padrão recorrente observado nos três modelos utilizados nesta etapa do experimento: concentração de predições na classe *NOISE* que concentra outliers e pontos fora dos clusters determinados, além da baixa recuperação das outras classes originais de comentários.

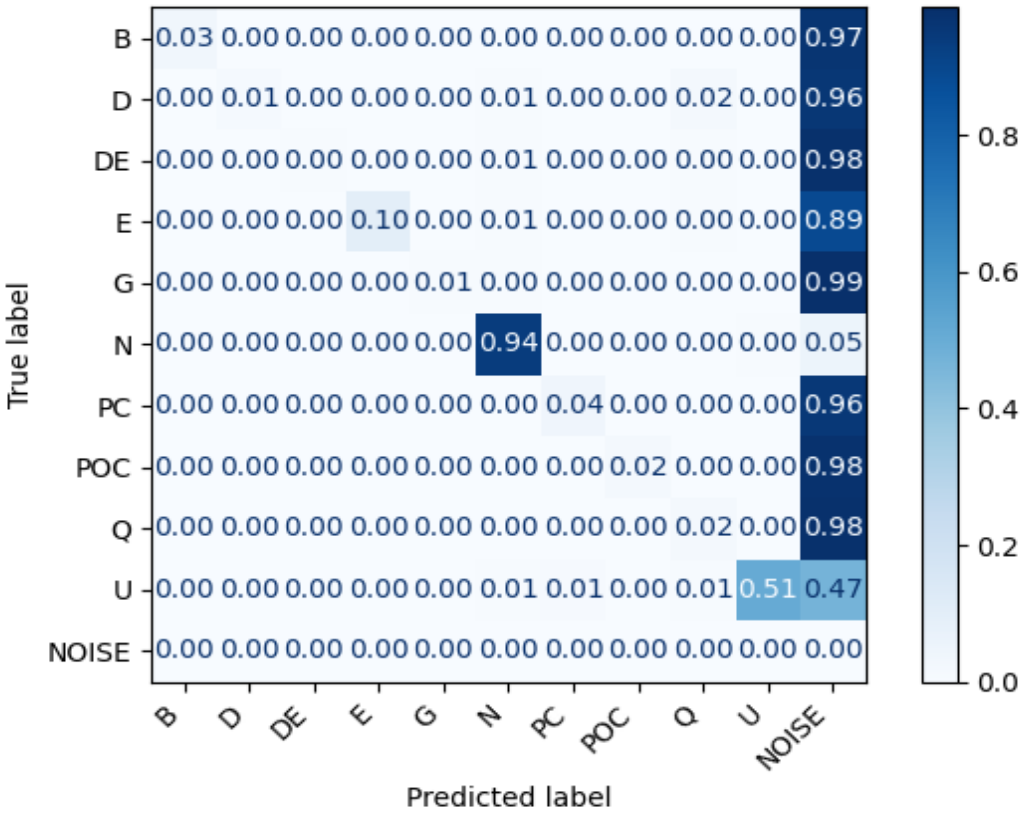


Figura 40: Matriz de confusão do modelo DBSCAN cenário (D-NF).

A matriz de confusão mostra que grande parte das instâncias de comentários, inclusive de classes frequentes como DE (*Description*), E (*Error*) e U (*Update*), foi absorvida pela categoria *NOISE*. Esse comportamento não é exclusivo do modelo *DBSCAN*, o modelo *HDBSCAN* também seguiu a mesma tendência, juntamente com o modelo *K-Means* que exibiu confusão elevada entre classes semanticamente próximas. Isso reforça a dificuldade dos métodos não supervisionados em identificar fronteiras claras quando operam sobre *embeddings* que apresentam forte sobreposição semântica entre categorias.

É importante destacar que, diferentemente dos modelos supervisionados, os modelos não supervisionados não produzem diretamente rótulos correspondentes às classes da taxonomia proposta. Dessa forma, a matriz de confusão apresentada não representa uma confusão no sentido clássico, mas sim um alinhamento entre os clusters gerados pelo modelo e as classes reais obtidas na rotulação manual, realizado exclusivamente para fins de análise e interpretação. Esse alinhamento permite observar como os agrupamentos identificados pelos modelos não supervisionados se relacionam com as classes conhecidas, sem implicar que os clusters correspondam diretamente a essas classes.

Ao comparar os resultados dos modelos não supervisionados com aqueles obtidos pelos modelos supervisionados, observa-se que as classes identificadas de forma consistente não são as mesmas**.** Enquanto os modelos supervisionados conseguem discriminar adequadamente classes como *Discussion*, *Update*, *Error* e *Guideline*, os modelos não supervisionados tendem a agrupar comentários de diferentes classes semânticas em clusters amplos ou classificá-los como *NOISE*. Isso indica que os agrupamentos formados refletem similaridades superficiais nos *embeddings* textuais, e não necessariamente as distinções funcionais capturadas pela taxonomia proposta. Dessa forma, os modelos não supervisionados não recuperam explicitamente as mesmas classes, mas evidenciam padrões latentes mais gerais, reforçando seu papel exploratório no contexto deste estudo.

Em conjunto, os resultados dos modelos não supervisionados apresentados neste subcapítulo confirmam que, apesar de úteis como ferramentas exploratórias, os modelos não supervisionados não foram capazes de recuperar a estrutura semântica do dataset

de comentários. Isso ocorre tanto por limitações teóricas dos modelos quanto pela natureza complexa dos comentários de relatórios de bugs, que apresentam ambiguidade semântica, alto desbalanceamento e forte sobreposição entre classes.

7.3 LLMs

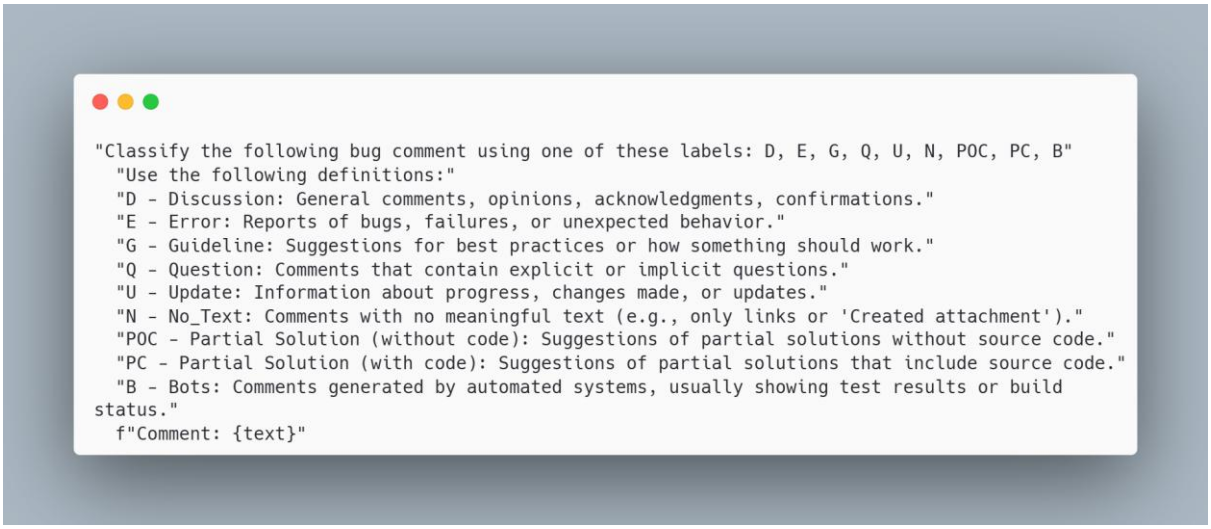
Neste subcapítulo são apresentados os resultados de uma etapa exploratória que investigou o uso de *Large Language Models* (LLMs) para a classificação automática de comentários de relatórios de bugs. Diferentemente das etapas anteriores, que envolveram modelos supervisionados e não supervisionados treinados ou ajustados diretamente sobre o *dataset*, esta etapa teve caráter exploratório, com o objetivo de avaliar o potencial de modelos de linguagem pré-treinados, utilizados via *prompting*, sem treinamento adicional, para capturar relações semânticas entre comentários e as classes definidas na taxonomia proposta.

Foram avaliados o modelo T5 (Text-To-Text Transfer Transformer) [53] e o Google Gemini [52], considerando sua capacidade de inferência contextual baseada em instruções textuais. A hipótese inicial era que modelos desse tipo poderiam superar limitações observadas nos modelos tradicionais, especialmente em classes semanticamente complexas ou minoritárias, devido à sua capacidade de generalização semântica e compreensão contextual. Diferentemente dos classificadores tradicionais avaliados nas seções anteriores, as LLMs possuem a capacidade de realizar inferência contextual com base em instruções textuais (*prompting*), o que as torna particularmente adequadas para tarefas envolvendo linguagem natural.

A interação com as LLMs foi realizada por meio de *prompting* estruturado, utilizando um prompt padronizado para guiar a classificação. O prompt incluía: a descrição completa das classes definidas na taxonomia; as instruções explícitas para que o modelo selecionasse apenas uma classe dentre o conjunto permitido; e exemplos ilustrativos de comentários previamente classificados, com seus respectivos rótulos, com o objetivo de orientar a inferência por similaridade.

A Figura 41 apresenta um exemplo do prompt utilizado durante os experimentos. Nesse prompt, o modelo recebe inicialmente uma instrução clara para classificar o

comentário de bug utilizando exclusivamente um conjunto pré-definido de rótulos. Em seguida, são fornecidas as definições textuais de cada classe da taxonomia, incluindo *Discussion* (D), *Error* (E), *Guideline* (G), *Question* (Q), *Update* (U), *No Text* (N), *Partial Solution Without Code* (POC), *Partial Solution With Code* (PC) e *Bot Comment* (B). Essas definições funcionam como contexto semântico para orientar o processo de inferência do modelo. Por fim, o comentário a ser classificado é inserido dinamicamente no final do prompt, permitindo que a LLM analise o conteúdo textual e produza um único rótulo correspondente à classe mais adequada segundo as instruções fornecidas.



```
"Classify the following bug comment using one of these labels: D, E, G, Q, U, N, POC, PC, B"
  "Use the following definitions:"
  "D - Discussion: General comments, opinions, acknowledgments, confirmations."
  "E - Error: Reports of bugs, failures, or unexpected behavior."
  "G - Guideline: Suggestions for best practices or how something should work."
  "Q - Question: Comments that contain explicit or implicit questions."
  "U - Update: Information about progress, changes made, or updates."
  "N - No_Text: Comments with no meaningful text (e.g., only links or 'Created attachment')."
  "POC - Partial Solution (without code): Suggestions of partial solutions without source code."
  "PC - Partial Solution (with code): Suggestions of partial solutions that include source code."
  "B - Bots: Comments generated by automated systems, usually showing test results or build
status."
  f"Comment: {text}"
```

Figura 41: Exemplo de prompt utilizado durante o experimento com LLMs.

Google Gemini

O modelo Gemini 2.0 Flash foi utilizado para classificar automaticamente comentários de relatórios de bugs conforme a taxonomia proposta. Apesar das instruções explícitas fornecidas no prompt, o modelo apresentou limitações significativas na aderência ao formato de resposta esperado. O Gemini frequentemente retornou texto adicional, cabeçalhos explicativos ou descrições em linguagem natural, o que resultou em desalinhamento entre o número de comentários fornecidos como entrada e o número de rótulos efetivamente produzidos.

Para cada inferência, foi fornecido ao modelo exclusivamente o texto individual do comentário a ser classificado, acompanhado das instruções e definições das classes. Não foi incluído contexto adicional da thread completa ou do histórico do relatório, com o objetivo de manter comparabilidade com os modelos supervisionados treinados sobre comentários individuais. Mesmo após a aplicação de filtros para extrair apenas rótulos válidos, o desempenho quantitativo permaneceu bastante baixo, com acurácia de 0.07 e F1 Macro de 0.05.

Observou-se ainda que classes como G (*Guideline*), PC (*Partial Solution*) e Q (*Question*) não foram corretamente preditas em nenhum comentário. Houve forte concentração de predições em poucas classes, principalmente E (*Error*) e N (*No Text*), indicando viés do modelo em direção a categorias mais genéricas. Esses resultados sugerem que, mesmo com instruções detalhadas e restrições explícitas, o modelo Gemini não conseguiu produzir classificações consistentes para o problema estudado.

T5 (Text-To-Text Transfer Transformer)

O modelo T5 foi avaliado sob o mesmo protocolo experimental adotado para o Gemini, utilizando *prompting* para classificar comentários em uma das classes da taxonomia proposta. Diferentemente do Gemini, entretanto, o comportamento do T5 mostrou-se ainda mais instável.

Os resultados qualitativos indicaram que o modelo frequentemente não respeitava o conjunto de rótulos permitido, mesmo com instruções explícitas. Foram observadas alucinações de classes inexistentes, combinações de múltiplas classes em uma única resposta e geração de texto livre fora do formato solicitado. Além disso, entradas idênticas resultaram em classificações diferentes em execuções distintas, evidenciando falta de consistência. Devido a essa instabilidade, não foi possível produzir métricas quantitativas confiáveis, como acurácia ou *F1-score*, tornando inviável uma comparação direta com os demais modelos avaliados neste estudo.

7.3.1 Limitações do Experimento com LLMs e Planos Futuros

Os experimentos exploratórios com uso dos modelos do T5 e Google Gemini evidenciam tanto o potencial quanto as limitações práticas do uso de LLMs para a classificação de comentários de relatórios de bugs seguindo classes específicas de comentários. As principais limitações observadas nesse contexto podem ser sintetizadas nos seguintes pontos:

- Aderência fraca ao conjunto de classes de comentários: Mesmo com a lista de classes explicitamente fornecida em todos os prompts, os modelos frequentemente retornaram textos adicionais, "explicações" ou produziram rótulos fora do conjunto esperado;
- Desempenho quantitativo insatisfatório: No caso do Gemini, mesmo após filtragem estrita para manter apenas rótulos válidos, as métricas foram baixas (Acurácia 0.07 e F1 Macro 0.05), com forte confusão entre classes; No T5, a inconsistência de formato e a alucinação de rótulos inviabilizaram uma avaliação quantitativa robusta;
- Dependência de engenharia de prompts e tempo de desenvolvimento: Nesse caso, ambos os modelos exigiram ciclos longos de ajuste de prompts; explorações sistemáticas para ajustar os formatos de saída, as descrições de classes e testes sucessivos para tentar alcançar estabilidade. Esse esforço de *prompt engineering* extrapola o tempo disponível no escopo desta pesquisa.

8 Resultados do Experimento de Previsão da Duração de Resolução de Relatórios de bugs

Neste capítulo apresentamos os resultados obtidos e a análise aprofundada dos experimentos de previsão da duração de resolução de relatórios de bugs, conduzidos por meio de modelos de machine learning para regressão e classificação, com o objetivo de prever a duração de resolução dos relatórios utilizando exclusivamente informações iniciais. Consideramos como informações iniciais os metadados básicos dos relatórios de bugs, os primeiros comentários e as primeiras mudanças de campos registrada no histórico de modificações dos relatórios de bugs.

Esta etapa experimental está diretamente relacionada às QP2 e QP3. Enquanto a QP2 investiga de que forma as classificações dos comentários influenciam a identificação do tempo necessário para a resolução dos relatórios de bugs, a QP3 amplia essa análise ao incorporar também as mudanças no histórico de modificações como variáveis explicativas. Assim, os modelos de regressão e de classificação temporal apresentados nesta seção têm como objetivo avaliar o impacto isolado e combinado de características textuais e estruturais na predição da duração dos relatórios ainda em seus estágios iniciais.

Os resultados e a avaliação são mostrados em duas frentes nesse capítulo: a primeira com resultados dos modelos utilizados para Regressão, o intuito dessa etapa do experimento era prever diretamente o número de dias até a resolução do relatório de bug. A segunda, com resultados dos modelos utilizados para classificação temporal, com o intuito de classificar o relatório em classes de duração, onde foram firmados dois cenários: um cenário com duas classes temporais de relatórios para a identificação de relatórios rápidos e lentos; e outro com quatro classes temporais de relatórios para a identificação de relatórios rápidos (*fast*), curto-prazo (*short-term*), médio prazo (*medium-term*) e longos (*long-term*). Nessa etapa do experimento também foram utilizados ajustes de hiperparâmetros com o auxílio da técnica *GridSearch*, e validação cruzada nos cenários para compreender o potencial e os limites de cada um dos modelos utilizados.

8.1 Modelos de Regressão

Este subcapítulo mostra os resultados e análises acerca dos resultados da etapa do experimento de previsão da duração de resolução de relatórios de bugs com o uso de modelos de regressão. Foram utilizados os modelos *Random Forest Regressor* [40], *XGBoost* [60] e *LightGBM* [61]. A escolha desses modelos foi motivada por sua capacidade de capturar relações não lineares, interações complexas entre atributos e distribuições assimétricas, características inerentes aos dados utilizados neste estudo, que combinam informações estruturadas dos relatórios de bugs, comentários textuais classificados, presença de anexos e histórico de modificações iniciais.

Tabela 13: Métricas de desempenho dos modelos de Regressão

Modelo	MAE	RMSE	R^2
Random Forest Regressor	329.9	592.6	0.12
XGBRegressor	340.5	600.5	0.09
LGBMRegressor	343.8	596.3	0.11

A Tabela 13 apresenta os valores obtidos para as métricas MAE (*Mean Absolute Error*), RMSE (*Root Mean Squared Error*) e R^2, utilizadas para avaliar o desempenho preditivo dos modelos de regressão. Os resultados evidenciam um padrão consistente entre os três modelos, indicando desempenho limitado para a tarefa de prever, em dias, a duração exata da resolução dos relatórios de bugs a partir de informações iniciais. O valor de MAE em torno de 330 dias indica que, em média, as previsões dos modelos apresentam um erro absoluto de aproximadamente 11 meses. Considerando que o experimento utiliza o mesmo *dataset* de relatórios de bugs empregado nas etapas anteriores, restringindo-se apenas aos relatórios cujos comentários foram previamente classificados, esse valor evidencia o alto grau de variabilidade temporal presente no processo de resolução de bugs.

Além disso, o RMSE próximo a 600 dias reforça essa interpretação, indicando a ocorrência de erros ainda mais elevados em casos específicos, o que sugere a presença de outliers temporais e forte assimetria na distribuição da variável alvo. Esse comportamento é esperado, uma vez que, durante o processo de coleta dos dados, não foi imposta nenhuma restrição quanto ao tempo máximo de resolução de um relatório, permitindo que bugs permaneçam abertos por períodos significativamente longos.

Por fim, os valores de R^2 variando entre 0.09 e 0.12 indicam que os modelos conseguem explicar aproximadamente 10% da variância total do tempo de resolução. Como o coeficiente de determinação varia no intervalo [0,1], valores próximos de zero indicam baixo poder explicativo, enquanto valores mais elevados, tipicamente acima de 0.5 em tarefas regressivas mais controladas, são considerados desejáveis. Nesse contexto, os resultados obtidos sugerem que a maior parte da variabilidade do tempo de resolução não pode ser explicada apenas pelas informações iniciais dos relatórios de bugs.

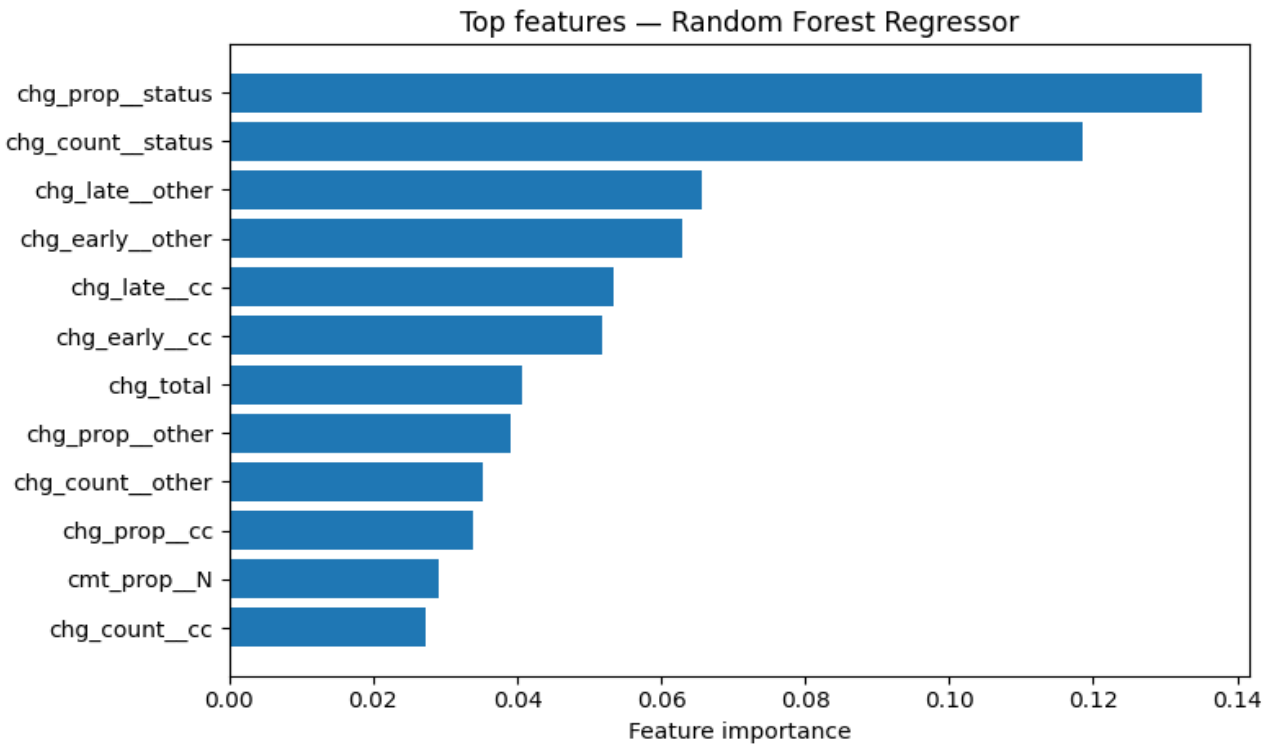


Figura 42: Importância das features no modelo Random Forest Regressor para a tarefa de previsão da duração de resolução de relatórios de bugs.

Apesar do desempenho quantitativo limitado, a análise das importâncias das features fornece indícios relevantes sobre padrões associados à duração dos relatórios. É importante ressaltar que os valores de importância das features variam de acordo com o modelo utilizado. No *Random Forest Regressor* e no *XGBoost*, as importâncias são normalizadas, assumindo valores no intervalo [0,1], permitindo a interpretação direta da contribuição relativa de cada atributo no modelo. Já no modelo *LightGBM*, a importância é baseada no número absoluto de divisões (*splits*) em que cada feature é utilizada ao longo das árvores, resultando em valores não normalizados.

A análise das importâncias das features do modelo *Random Forest Regressor*, apresentada na Figura 42, evidencia que os atributos mais relevantes estão fortemente associados à dinâmica administrativa precoce dos relatórios de bugs. As features *"chg_prop__status"* e *"chg_count__status"* figuram como as mais importantes, indicando que tanto a proporção quanto a frequência de mudanças de status nos estágios iniciais do relatório possuem forte correlação com o tempo total de resolução. Além disso, atributos como *"chg_early__other"*, *"chg_late__other"* e *"chg_early__cc"* reforçam que alterações frequentes em campos administrativos e na lista de colaboradores (CC), especialmente no início do ciclo de vida do bug, refletem instabilidade no entendimento do problema. Esse padrão sugere que relatórios que demandam múltiplas revisões administrativas precoces tendem a apresentar maior complexidade percebida, o que se traduz em tempos de resolução mais longos.

A Figura 43 apresenta as principais features identificadas pelo modelo *XGBoost Regressor.* Observa-se que as mudanças relacionadas ao status (*"chg_count__status"* e *"chg_prop__status"),* continuam sendo os atributos mais relevantes, corroborando os achados do Random Forest. Adicionalmente, o modelo atribui importância significativa à features associadas a tipos específicos de comentários nos estágios iniciais e intermediários do relatório, como "*early__G*", "*early__Q*", "*early__POC*" e "mid__POC". Essas features estão relacionadas, respectivamente, a diretrizes, questionamentos e soluções parciais, indicando que a presença precoce de dúvidas, orientações técnicas e tentativas iniciais de solução está associada a um aumento no tempo de resolução. Esse comportamento sugere que relatórios que demandam maior

esclarecimento conceitual ou experimentação inicial tendem a ser mais complexos e demorados.

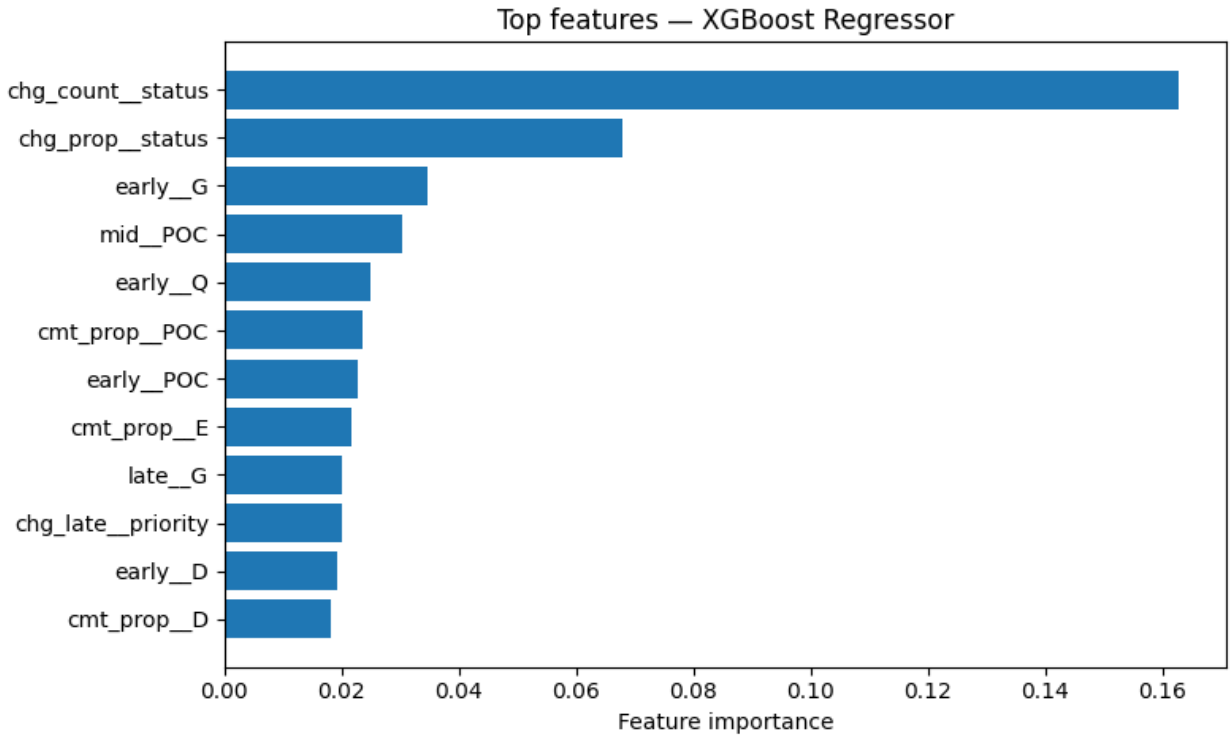


Figura 43: Importância das principais features identificadas pelo modelo XGBoost Regressor para a predição do tempo de resolução de relatórios de bugs.

A análise das importâncias das features do modelo *LightGBM Regressor*, apresentada na Figura 44, reforça o papel central da atividade volumétrica precoce na determinação do tempo de resolução dos relatórios de bugs. As features mais relevantes (*"chg_early__other"*, *"chg_late__other"*, *"chg_early__cc"* e *"chg_late__cc"*) indicam que tanto o volume quanto a distribuição temporal das modificações nos campos do relatório influenciam significativamente a duração do processo de resolução. Diferentemente dos modelos anteriores, o *LightGBM* enfatiza fortemente o número absoluto de interações e modificações, refletido pelo alto número de splits associados a essas features. Esse resultado sugere que relatórios caracterizados por múltiplas alterações, especialmente fora dos campos mais estruturados, tendem a apresentar maior ambiguidade e maior esforço de coordenação, impactando negativamente o tempo de resolução.

No contexto da QP2, os resultados de regressão indicam que as variáveis derivadas das classificações dos comentários contribuem para explicar parte da variabilidade do tempo de resolução. Embora os atributos estruturais apresentem maior poder explicativo isolado, a inclusão das características textuais proporciona ganho incremental consistente, evidenciando que o perfil comunicacional exerce influência complementar na dinâmica de resolução dos relatórios.

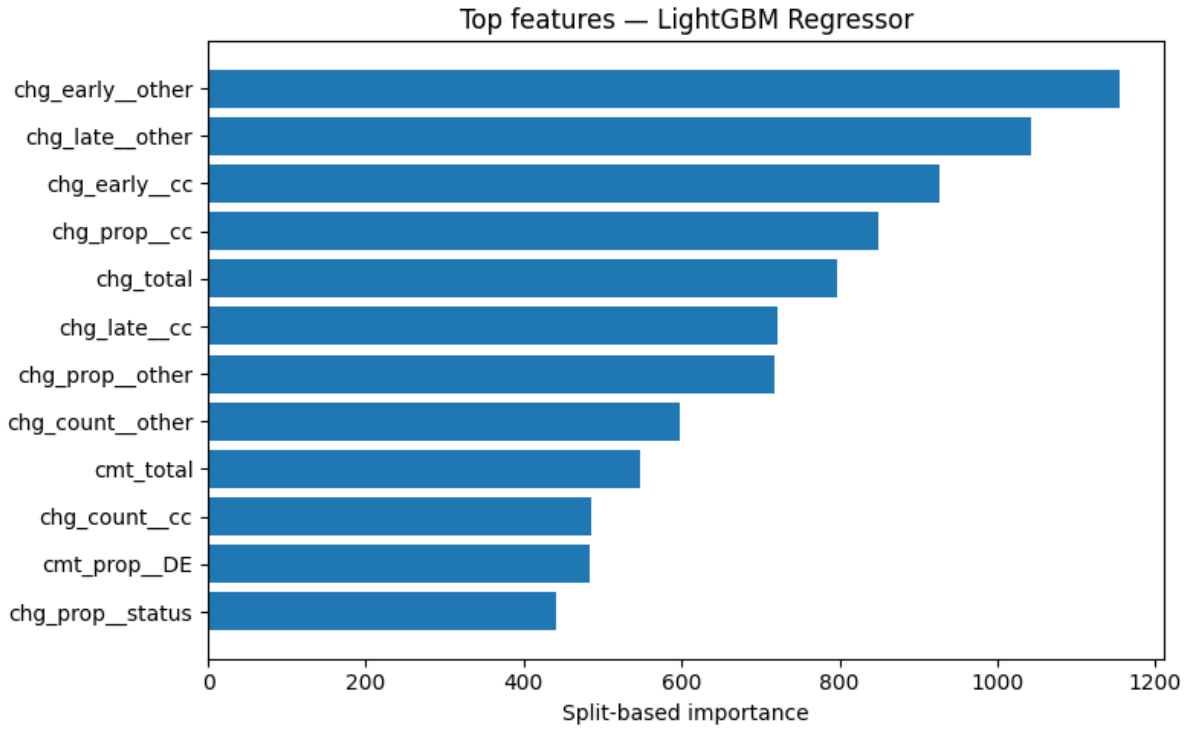


Figura 44: Importância das principais features selecionadas pelo modelo LightGBM Regressor para a predição do tempo de resolução de relatórios de bugs.

8.2 Modelos de Classificação Supervisionada

Este subcapítulo mostra os resultados dos modelos supervisionados utilizados para classificação, com o intuito de classificar o relatório em classes de duração, onde foram firmados dois cenários: um cenário com duas classes temporais, para a identificação de relatórios rápidos e lentos; e outro com quatro classes temporais, para a identificação de relatórios rápidos (*fast*), de curto-prazo (*short-term*), de médio prazo (*medium-term*) e longos (*long-term*). Nessa etapa do experimento também foram utilizados ajustes de

hiperparâmetros com o auxílio da técnica *GridSearch*. Em ambos os casos, as features utilizadas combinam dois tipos complementares de informação. O primeiro corresponde a sinais estruturais, extraídos do comportamento inicial do relatório de bug, como a dinâmica de mudanças em campos administrativos, o volume e a frequência de alterações registradas e a intensidade de interação formal ao longo das primeiras etapas do ciclo de vida do relatório. Esses sinais refletem o grau de instabilidade, retrabalho e complexidade percebida logo no início do processo de triagem. O segundo grupo corresponde a sinais semânticos, derivados da classificação dos comentários, representando o perfil textual e comunicacional associado ao relatório, como a predominância de discussões, descrições técnicas, atualizações, perguntas ou tentativas de solução. A combinação desses dois grupos de sinais permite que os modelos capturem simultaneamente aspectos organizacionais e comunicacionais do relatório, fornecendo uma visão mais abrangente para a classificação das classes temporais de duração.

Após a definição das classes temporais, foram treinados e avaliados diferentes modelos de classificação supervisionada, são eles: *Decision Tree*, *Random Forest*, *Logistic Regression*, *LinearSVC*, *SVM com kernel RBF* e *Multinomial Naive Bayes*. Esses modelos foram aplicados de forma consistente em ambos os cenários de classificação temporal (duas e quatro classes), sendo todos submetidos a um processo de ajuste de hiperparâmetros por meio da técnica de *GridSearch*, com o objetivo de identificar as melhores configurações para cada modelo e cenário avaliado.

Classificação com duas classes temporais

Esse é o cenário de classificação onde consideramos duas classes temporais para identificar se os relatórios de bugs são rápidos ou lentos. Por rápidos, consideramos uma janela de trinta dias até o final do ciclo de resolução dos relatórios de bugs. Consideramos o relatório como lento, se o tempo para o final do ciclo de resolução for acima de trinta dias.

A definição da janela de 30 dias foi estabelecida após análise exploratória da distribuição do tempo de resolução no dataset, que indicou uma concentração significativa de relatórios resolvidos nesse intervalo. Esse ponto de corte permite

distinguir relatórios resolvidos dentro de um período operacionalmente considerado curto das demais ocorrências, preservando equilíbrio razoável entre as classes no cenário binário.

Tabela 14: Resultados da classificação em duas classes temporais

Modelo	**Acurácia**	**Precisão**	**Revocação**	**F1 Macro**
LinearSVC	0.865	0.864	0.865	0.864
SVM RBF	0.863	0.862	0.862	0.862
Logistic Regression	0.862	0.861	0.861	0.861
Random Forest	0.862	0.861	0.861	0.861
Decision Tree	0.858	0.856	0.857	0.857
Multinomial Naive Bayes	0.709	0.729	0.692	0.690

A Tabela 14 mostra os resultados da previsão da duração de resolução de relatórios de bugs em duas classes temporais através das métricas de Acurácia, Precisão, Revocação e F1 Macro. Com exceção do modelo *Naive Bayes Multinomial*, os demais modelos utilizados apresentaram um desempenho alto no cenário binário (rápido ou lento), convergindo para valores de acurácia próximos de 0.86, acompanhados de precisão, Revocação e F1 Macro igualmente equilibrados. Esse equilíbrio de valores é extremamente importante pois mostra que os classificadores não apenas acertam na média, mas possuem capacidades para identificar com a mesma eficácia tanto os relatórios com tempo de resolução rápido, quanto os lentos, fato esse comprovado particularmente nos valores de Revocação acima de 0.86. Com base nesses resultados, observa-se que o problema de distinguir, a partir de informações iniciais, se um relatório de bug será resolvido de forma rápida ou lenta apresenta boa separabilidade. Isso significa que existe um conjunto de sinais precoces suficientemente informativo para permitir que diferentes famílias de modelos lineares, baseados em margens máximas e baseados em árvores, aprendam padrões semelhantes e atinjam níveis de desempenho próximos entre si.

Dentre os seis modelos utilizados nesta fase do experimento, o que obteve o melhor desempenho foi o modelo *LinearSVC*. A Tabela 15 detalha o desempenho por classe para o melhor modelo. As métricas mostram um comportamento bem equilibrado no modelo *LinearSVC*: bugs rápidos (classe 0) e bugs lentos (classe 1) são recuperados com níveis praticamente idênticos de precisão, revocação e F1-score, o que indica ausência de viés forte em favor de uma das classes. O modelo *Decision Tree* também apresentou desempenho competitivo, com acurácia de 0,858 e F1 Macro de 0,857, valores muito próximos aos obtidos pelos demais modelos. O processo de ajuste de hiperparâmetros contribuiu para reduzir efeitos de sobreajuste, permitindo que o modelo mantivesse boa capacidade de generalização mesmo sendo mais simples em sua estrutura de decisão.

Nesse cenário binário, a proximidade entre *precision* e *recall* reforça que o bom desempenho não se deve apenas ao acerto médio, mas também a um equilíbrio efetivo entre confiabilidade e cobertura das classes. No melhor caso, o LinearSVC alcança acurácia de 0,865, *precision* de 0,864 e *recall* de 0,865, indicando que o modelo consegue tanto gerar previsões consistentes quanto recuperar, de forma homogênea, relatórios rápidos e lentos. No contexto de manutenção de software, esse equilíbrio é importante porque um *recall* alto para relatórios lentos reduz o risco de deixar passar casos potencialmente problemáticos, enquanto uma *precision* alta evita alertas excessivos e mantém a utilidade prática da previsão para triagem e planejamento.

Tabela 15: Métricas por classe temporal do modelo LinearSVC

Classe Temporal	Precisão	Revocação Macro	F1 Macro
0 (rápido)	0.88	0.87	0.88
1 (lento)	0.85	0.86	0.85

Para aprofundar a interpretação dos modelos, foram analisadas as importâncias das features nos classificadores baseados em árvores, especificamente o *Random Forest* e o *Decision Tree*, por serem os únicos modelos deste cenário que fornecem suporte direto à extração de importância de atributos. As Figuras 44 e 45 apresentam,

respectivamente, as features mais relevantes segundo o modelo *Random Forest* e o *Decision Tree* no cenário de classificação com duas classes temporais.

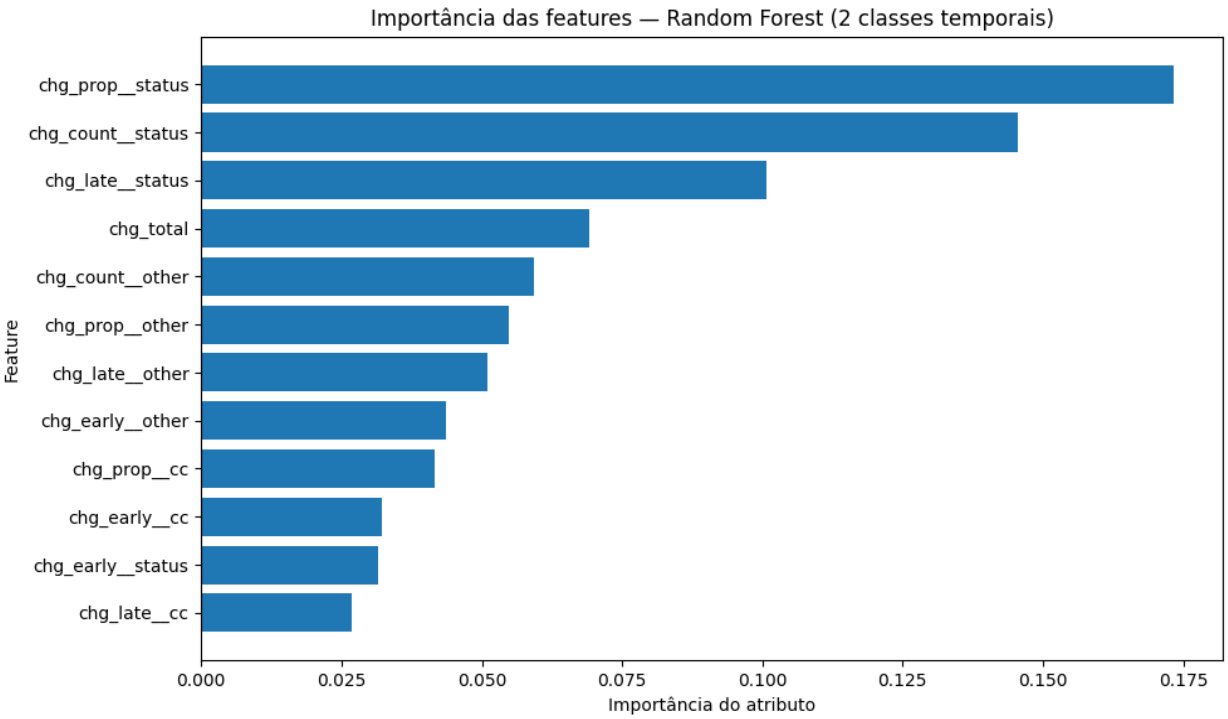


Figura 45: Importância das features no modelo Random Forest para classificação em duas classes temporais.

A análise das importâncias das features revela que os atributos mais relevantes estão fortemente associados à dinâmica precoce de instabilidade do relatório de bug. Entre as principais features destacam-se "*chg_prop__status*", que representa a proporção de mudanças no campo *status*; "*chg_count__status*", que captura a quantidade absoluta dessas mudanças; e "*chg_late__status*", relacionada às alterações de status próximas ao final da janela inicial analisada. Além disso, atributos como "*chg_total*", "*chg_count__other*" e "*chg_prop__other*" indicam o volume global de alterações em campos administrativos, enquanto "*chg_early__cc*" e "*chg_late__cc*" refletem a dinâmica de inclusão e remoção de participantes (campo *CC*) ao longo do início do ciclo de vida do relatório.

Embora a hipótese inicial considerasse que sinais derivados da classificação dos comentários poderiam desempenhar papel central na previsão da duração dos relatórios, a análise de importância de atributos revelou que variáveis estruturais associadas à dinâmica de mudanças administrativas tiveram maior peso discriminativo no cenário binário. Esse resultado dialoga diretamente com a QP3, ao evidenciar que as mudanças no histórico de modificações, em especialmente alterações frequentes de status, movimentações no (campo CC) e volume global de mudanças nos estágios iniciais, apresentam forte associação com o tempo de resolução dos relatórios.

Em relação à QP2, os atributos derivados das classificações dos comentários também demonstraram relevância, ainda que de forma complementar. A presença e predominância de determinados tipos de comentários, como *Discussion*, *Question* ou *Partial Solution*, contribuem para caracterizar o perfil comunicacional do relatório e auxiliam na distinção entre casos limítrofes. Em conjunto, os resultados indicam que o tempo de resolução não é determinado exclusivamente pela natureza semântica das interações nem apenas pela dinâmica estrutural das modificações, mas pela combinação desses dois grupos de sinais, sendo os sinais organizacionais precoces particularmente mais influentes na tarefa binária avaliada.

Ainda em relação à QP2, os resultados obtidos com modelos de classificação supervisionada reforçam que as características derivadas da classificação dos comentários apresentam menor poder discriminativo quando utilizadas isoladamente para prever a duração dos relatórios. No entanto, sua inclusão em conjunto com variáveis estruturais do histórico de modificações resulta em melhora incremental consistente nas métricas de desempenho. Esse comportamento evidencia que, embora os comentários não sejam os principais determinantes do tempo de resolução, eles atuam como sinal complementar relevante no processo de identificação de relatórios com maior ou menor duração.

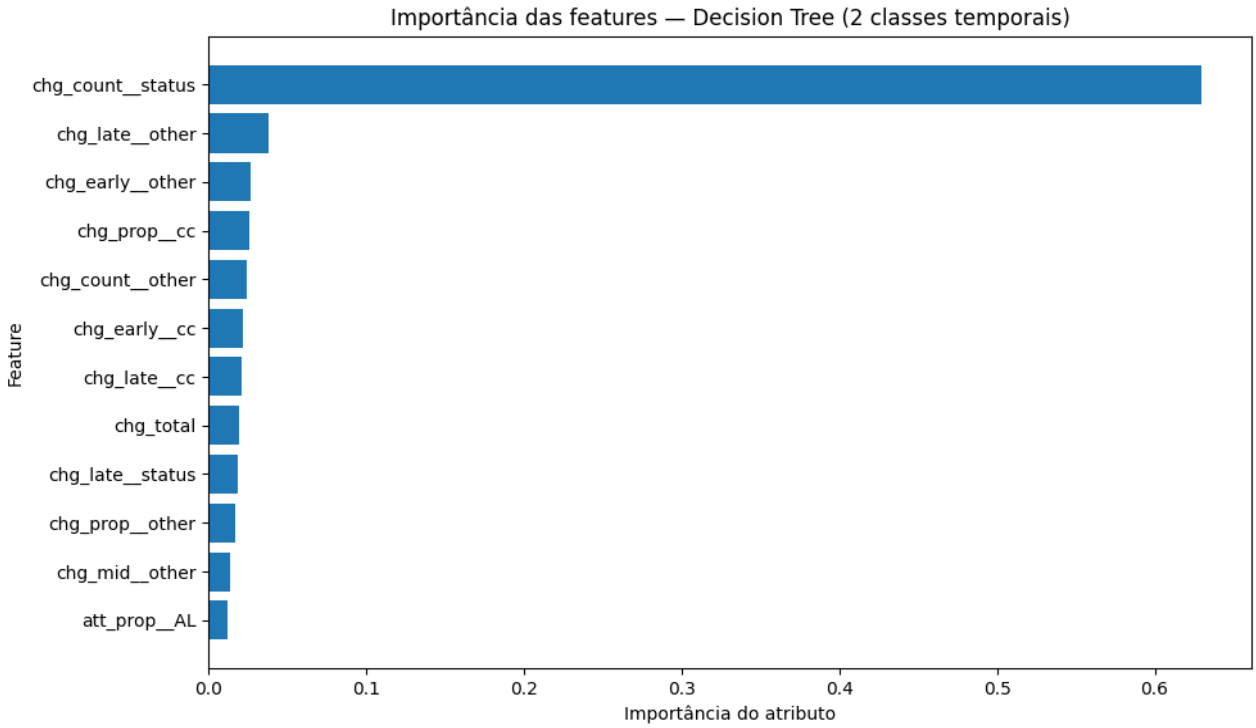


Figura 45: Importância das features no modelo Decision Tree para classificação em duas classes temporais.

Essas features estão associadas a padrões de instabilidade precoce no relatório de bug, caracterizados por múltiplas mudanças de status, alterações frequentes no campo CC e volume elevado de modificações nos primeiros dias. Estatisticamente, esses padrões mostraram correlação com a classe de maior duração. Uma possível interpretação é que relatórios que exigem retrabalho inicial, redefinição de responsáveis ou revisões frequentes de estado refletem maior complexidade técnica ou organizacional, o que pode prolongar o tempo até a resolução. É importante destacar que essa análise identifica associação estatística e não estabelece relação causal direta.

Em contrapartida, bugs rápidos tendem a apresentar históricos bem mais lineares: menos mudanças de status, poucas alterações de CC, e, em muitos casos, um caminho de resolução mais direto. Ao combinar esses sinais estruturais com estatísticas derivadas da classificação dos comentários, os modelos passam a capturar não apenas a intensidade da interação, mas também a natureza semântica dessa interação nos primeiros dias do relatório. A presença recorrente de comentários classificados como

Discussion e *Question* indica que o problema ainda está em processo de compreensão coletiva, enquanto comentários do tipo *Guideline* e *Partial Solution* sugerem tentativas iniciais de direcionamento ou resolução parcial. Quando esses padrões aparecem associados a múltiplas mudanças de status e alterações frequentes em CC, formam um sinal composto de incerteza e retrabalho precoce, que se mostrou altamente correlacionado com tempos de resolução mais longos. Essa combinação explica o alto e estável desempenho observado na separação binária entre bugs rápidos e lentos.

Vale destacar que modelos como *Logistic Regression, LinearSVC*, *SVM* e *Multinomial Naive Bayes* não fornecem medidas diretas de importância de atributos comparáveis às obtidas em modelos baseados em árvores. Essa limitação restringe análises interpretativas mais detalhadas para esses classificadores, mas também aponta uma oportunidade para trabalhos futuros, nos quais técnicas de interpretabilidade modelo-agnósticas, como *SHAP* [62] ou *LIME* [63], podem ser exploradas para aprofundar a análise das features.

Classificação com quatro classes temporais

Nesta etapa do estudo, conduzimos o experimento com os modelos supervisionados tendo como objetivo de prever, logo no início da vida de um relatório, quanto tempo provavelmente será necessário para sua resolução. Diferentemente da classificação binária anterior, aqui buscamos capturar quatro classes distintas de duração. A definição das quatro classes não foi arbitrária, antes da execução, realizamos uma análise detalhada da distribuição dos tempos de resolução dos relatórios de bugs presentes no *dataset*, observando que os valores apresentam um comportamento assimétrico: grande concentração de resoluções rápidas nos primeiros dias, seguida por uma longa quantidade de casos complexos, reabertos ou de difícil reprodução. Com base nessa análise da distribuição, definimos os seguintes limites:

- classe *fast*, até 7 dias para a resolução;
- classe *short_term*, entre 7 e 30 dias para a resolução;
- classe *medium_term*, entre 30 e 180 dias para a resolução;

- classe *long_term*, acima de 180 dias para a resolução.

A Figura 47 mostra uma representação dos relatórios de bugs resultante na escolha dos intervalos de tempo de cada uma das classes. Ela evidencia que, embora exista uma assimetria natural, os intervalos temporais (*buckets*) definidos para cada classe produzem uma estrutura de dados mais equilibrada, com valores quantitativos de relatórios de bugs de 3445 relatórios de bugs na classe "*fast*"; 1600 relatórios de bugs na classe "*short_term*"; 1550 relatórios de bugs na classe "*medium_term*"; e 2635 relatórios de bugs na classe "*long_term*". Neste contexto, o termo *bucket* refere-se a um intervalo de tempo discreto utilizado para agrupar relatórios de bugs de acordo com sua duração de resolução. Essa escolha permitiu treinar os modelos sem viés extremo e aumentou a capacidade de avaliar corretamente o comportamento das classes intermediárias.

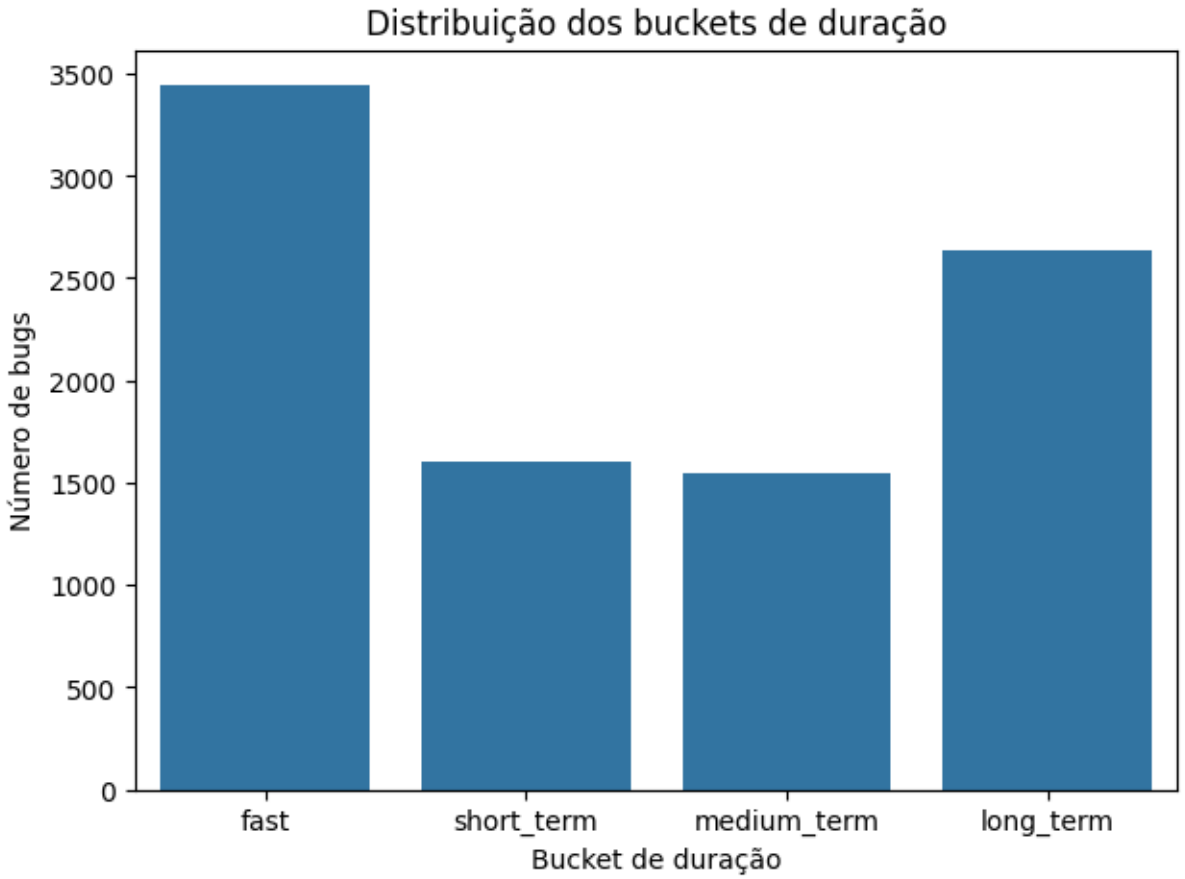


Figura 47: Distribuição dos relatórios de bugs por classes de duração temporal.

Os resultados gerais mostraram que, embora nenhum modelo tenha alcançado desempenho elevado, todos foram capazes de capturar parcialmente os padrões de duração presentes nos dados. A Tabela 16 mostra em detalhes os resultados dos modelos de classificação com quatro classes temporais. O melhor desempenho global foi obtido pelo *LinearSVC*, com acurácia de 0.643 e métricas macros superiores às demais alternativas. O *SVM* com kernel RBF obteve resultados similares, especialmente no Revocação macro, indicando capacidade mais equilibrada de recuperação das quatro classes. *Random Forest* e *Decision Tree* apresentaram desempenhos intermediários, enquanto o *Naive Bayes*, como esperado para dados de alta dimensionalidade e distribuição altamente desbalanceada, obteve os piores resultados.

Tabela 16: Resultados da classificação com quatro classes temporais

Modelo	Acurácia	Precisão	Revocação	F1 Macro
LinearSVC	0.643	0.561	0.537	0.525
Logistic Regression	0.633	0.548	0.534	0.525
SVM RBF	0.621	0.548	0.534	0.542
Random Forest	0.622	0.529	0.519	0.509
Decision Tree	0.609	0.537	0.534	0.534
Naive Bayes Multinomial	0.460	0.425	0.403	0.400

Mesmo entre os melhores modelos, o comportamento por classe revela uma forte assimetria na capacidade preditiva. Em todos os modelos, as classes extremas "*fast"* e "*long_term"*, foram significativamente mais fáceis de identificar.

Essa capacidade de identificar com maior precisão as classes extremas possui implicações práticas relevantes no contexto de manutenção de software. A previsão precoce de relatórios *"long_term"* pode auxiliar equipes de desenvolvimento a priorizar recursos, redistribuir tarefas ou escalar problemas complexos logo nos estágios iniciais, reduzindo o risco de prolongamento excessivo. Da mesma forma, a identificação confiável de relatórios "fast" permite que equipes reconheçam casos de baixa complexidade e os tratem de forma mais ágil, evitando alocação desnecessária de esforço especializado.

Em termos organizacionais, essa distinção pode apoiar decisões de triagem automatizada, planejamento de sprint e estimativas de carga de trabalho, especialmente em ambientes com grande volume de relatórios. Assim, mesmo que a classificação em quatro classes apresente limitações nas categorias intermediárias, a robustez na identificação das classes extremas já oferece um mecanismo potencial de suporte à gestão do processo de resolução de bugs.

A Tabela 17 mostra as métricas detalhadas para o modelo LinearSVC, que melhor desempenhou dentre os demais modelos desta etapa do estudo. Para a classe *"fast"*, o modelo *LinearSVC* alcançou Revocação a um valor de 0.92 e com F1 acima de 0.80, evidenciando que os relatórios resolvidos rapidamente apresentam padrões iniciais consistentes com baixa oscilação de status, menor número de envolvidos e reduzido volume de comentários, conforme indicado pelas features estruturais mais relevantes atribuídas pelos modelos baseados em árvores, como proporção e contagem de mudanças de status, volume total de alterações e número de participantes em CC. De modo semelhante, a classe "*long_term"* também apresentou desempenho consistentemente superior, com Revocação com valor de 0.78 e F1 acima de 0.70. Essa previsibilidade ocorre porque relatórios que tendem a se prolongar geralmente mostram, ainda nos estágios iniciais, uma combinação de redundâncias, tentativas de reprodução

malsucedidas, entrada de novos participantes na discussão e mudanças repetidas de prioridade ou estado, padrões esses facilmente capturados por métricas como *"chg_prop__status"*, *"chg_count__status"*, *"chg_total"* e *"chg_early__cc"*.

Tabela 17: Métricas por classe temporal do modelo LinearSVC

Classe Temporal	Precisão	Revocação	F1 Macro
fast	0.73	0.92	0.81
short_term	0.42	0.19	0.27
medium_term	0.47	0.25	0.33
long_term	0.62	0.78	0.70

As classes intermediárias "*short_term"* (7–30 dias) e "*medium_term"* (30–180 dias), demonstraram desempenho substancialmente inferior, com F1 raramente ultrapassando 0.35, independentemente do modelo. A principal razão para essa dificuldade está na própria natureza dos relatórios pertencentes a essas faixas. Ao contrário das classes extremas, que possuem comportamentos bem definidos, os relatórios intermediários apresentam grande sobreposição semântica e processual, especialmente nos primeiros dias, período que compõe a janela de informações usada neste estudo. Em outras palavras, os sinais iniciais que distinguem um bug resolvido em dez dias de outro que levará noventa dias não são suficientemente marcados no início da vida do relatório, o que limita a capacidade discriminativa de modelos supervisionados tradicionais.

Na classificação em quatro classes, *precision* e *recall* tornam ainda mais evidente a assimetria do problema. Para a classe *fast*, o LinearSVC alcança *recall* de 0,92 e F1 de 0,81, mostrando alta capacidade de recuperar relatórios resolvidos rapidamente; para "*long_term"*, o *recall* de 0,78 e F1 acima de 0,70 indicam que a identificação precoce de casos prolongados também é relativamente robusta. Em contraste, as classes intermediárias apresentam *precision* e *recall* significativamente menores, o que mostra

que o modelo tem dificuldade tanto para reconhecer corretamente essas instâncias quanto para evitar confundi-las com outras faixas temporais. Nesse contexto, o *recall* é particularmente importante para a classe "*long_term"*, pois mede a capacidade de capturar relatórios com maior risco de atraso, enquanto a *precision* nessa mesma classe indica o quão confiáveis são esses alertas para uso gerencial e operacional.

Outro aspecto fundamental para a interpretação dos resultados é a análise das features mais importantes. Os modelos baseados em árvores atribuíram maior relevância a indicadores de instabilidade e complexidade processual, como a proporção e a frequência de mudanças de status (*"chg_prop__status"* e *"chg_count__status"*), o volume precoce de interações (*"chg_total"*, *"chg_count__other"*) e a entrada inicial de novos participantes na lista de CC (*"chg_early__cc"*). Esses atributos se alinham com a literatura que sugere que a dinâmica inicial de um relatório, e não apenas seu conteúdo textual, é um forte preditor de sua duração.

Por fim, os resultados indicam que a classificação da duração de bugs em quatro classes representa um problema extremamente desafiador quando baseada apenas nos sinais iniciais do relatório, que se referem ao conjunto de informações disponíveis nos estágios iniciais do ciclo de vida do relatório de bug, incluindo metadados básicos, histórico precoce de mudanças de campos (como status, prioridade e CC) e estatísticas derivadas dos primeiros comentários, sem considerar eventos ou interações posteriores ao período inicial analisado. Ainda assim, os modelos conseguem capturar adequadamente as classes extremas, revelando que a complexidade subjacente a essas categorias é visível desde o início. A dificuldade nas classes intermediárias, por sua vez, reforça limitações já identificadas em estudos anteriores e sugere a necessidade de abordagens mais sofisticadas, como modelos temporais, *embeddings* contextuais de comentários e integração posterior de dados evolutivos. Apesar dessas dificuldades, a análise obtida aqui contribui para um entendimento mais profundo das possibilidades e limitações do uso de aprendizado supervisionado na previsão precoce da duração de relatórios de bugs.

Em relação à QP3, os resultados demonstram que a combinação entre histórico de modificações e classificações dos comentários proporciona melhor capacidade discriminativa na tarefa de classificação temporal do que a utilização isolada de

qualquer um desses grupos de atributos. As variáveis estruturais associadas às mudanças iniciais no relatório apresentaram maior peso preditivo; contudo, a inclusão das características textuais capturadas pela classificação dos comentários permitiu identificar nuances comunicacionais que não são plenamente refletidas apenas nos metadados históricos.

9 Conclusões

Este capítulo apresenta a síntese final dos resultados obtidos ao longo deste estudo, consolidando as principais evidências empíricas produzidas e articulando-as às questões de pesquisa propostas. As conclusões são estruturadas de forma a integrar os achados das diferentes etapas experimentais, abrangendo a classificação de comentários, a análise da influência desses comentários na duração dos relatórios de bugs e a predição do tempo de resolução a partir de informações iniciais.

Além da discussão dos resultados à luz das questões de pesquisa, este capítulo também aborda as limitações do trabalho, destaca sua relevância prática e científica, e explicita as contribuições oferecidas à área de engenharia de software empírica. Por fim, são delineadas direções para trabalhos futuros, indicando oportunidades de aprofundamento e expansão das abordagens investigadas, especialmente no contexto da análise automatizada de comunicação textual e dinâmica de manutenção em sistemas de rastreamento de bugs.

9.1 Respostas às Questões de Pesquisa

O presente estudo investigou o papel dos comentários, da comunicação textual e das características estruturais dos relatórios de bugs na compreensão e previsão do comportamento desses artefatos ao longo do seu ciclo de vida. Para isso, três questões de pesquisa foram formuladas, cada uma associada a etapas específicas dos experimentos conduzidos no trabalho. A primeira, QP1: *"Como os comentários de relatórios de bugs podem ser classificados seguindo seu conteúdo textual, anexos e influência no processo de resolução de um relatório de bug?"*, buscou compreender como os comentários podem ser classificados automaticamente considerando seus diferentes papéis comunicativos e técnicos. A segunda, QP2: *"De que forma as características e classificações dos comentários influenciam na identificação do tempo necessário para a resolução dos relatórios de bugs?"*, investigou de que forma as características e classificações dos comentários influenciam a duração de resolução dos

relatórios de bugs. Por fim, a QP3: *"Como as mudanças no histórico de modificações dos relatórios de bugs e os tipos de comentários em relatórios de bugs se relacionam com o tempo de resolução do relatório de bug?"* questionou se, a partir de um conjunto amplo de características estruturais, históricas e textuais, é possível classificar os relatórios de bugs conforme o tempo estimado até sua resolução. As conclusões apresentadas a seguir integram os resultados obtidos em cada uma dessas etapas.

O primeiro experimento, de classificação de comentários, abordou exclusivamente a QP1, investigando a viabilidade e a acurácia da classificação automática de comentários de relatórios de bugs. Para isso, foi construída uma classificação de dez tipos de comentários, validada manualmente e aplicada a um conjunto de mais de 36 mil comentários classificados. O *dataset* mostrou-se altamente desbalanceado, com predominância de comentários do tipo *Bots*, *Description* e *Updates*, enquanto classes mais técnicas, como *Errors* e *Partial Solutions*, apresentavam baixa frequência. Esse contexto motivou a adoção de múltiplas técnicas de balanceamento, além de um extenso pipeline de pré-processamento textual (normalização, limpeza, tokenização e lematização). Os modelos supervisionados, especialmente *Random Forest* e *Logistic Regression*, demonstraram capacidade consistente de aprender padrões linguísticos e comunicativos presentes nos comentários, mesmo diante da distribuição assimétrica. Em contrapartida, modelos não supervisionados e LLMs não foram capazes de atender às exigências da tarefa, ora produzindo clusters incoerentes, ora desobedecendo às instruções rígidas da classificação. Assim, para a QP1, conclui-se que os comentários podem, de fato, ser classificados automaticamente de forma satisfatória, quando modelos supervisionados tradicionais sejam empregados em conjunto com técnicas adequadas de balanceamento e pré-processamento.

Em relação à QP2, os resultados indicam que as classificações dos comentários exercem influência na identificação do tempo de resolução, porém essa influência não se mostrou dominante quando comparada às variáveis estruturais. Os atributos derivados das classes de comentários contribuíram para enriquecer a modelagem, especialmente na distinção de casos limítrofes e na caracterização do perfil comunicacional dos relatórios. Comentários do tipo Update, Question e Partial Solution

mostraram associação com relatórios mais longos, sugerindo maior necessidade de interação técnica e retrabalho. Assim, conclui-se que as classificações dos comentários desempenham papel complementar e informativo na predição da duração dos relatórios, ampliando a capacidade explicativa dos modelos quando combinadas com sinais estruturais.

No que se refere à QP3, os resultados mostram que a relação entre histórico de modificações e tipos de comentários com o tempo de resolução é assimétrica. As variáveis estruturais associadas à instabilidade precoce do relatório, como frequência de mudanças de status, volume total de alterações e movimentação no (*campo CC*), apresentaram maior poder discriminativo na classificação temporal, especialmente no cenário binário. Entretanto, a inclusão das classificações de comentários proporcionou ganho incremental consistente, indicando que a natureza semântica das interações contribui para capturar nuances que não são totalmente refletidas nos metadados estruturais. Dessa forma, a predição do tempo de resolução se mostrou mais eficaz quando combinou sinais históricos e textuais, evidenciando que ambos os grupos de características se relacionam com a duração do relatório, ainda que com pesos distintos.

Do ponto de vista das métricas, a interpretação de *precision* e *recall* foi central para distinguir desempenho estatístico de utilidade prática. Enquanto a *precision* permitiu avaliar a confiabilidade das previsões emitidas pelos modelos, a *recall* mostrou o quanto relatórios potencialmente críticos, especialmente os de maior duração, foram efetivamente recuperados. Essa distinção é importante porque, em cenários reais de manutenção, um modelo útil não deve apenas acertar na média, mas também identificar precocemente os casos que exigem maior atenção da equipe. Conclui-se, portanto, que o tempo de resolução dos relatórios de bugs está mais fortemente associado à dinâmica estrutural inicial, mas é também influenciado pelo perfil comunicacional observado nos comentários, reforçando a natureza multifatorial do processo de resolução.

Este estudo mostra que a comunicação textual presente nos relatórios de bugs não é apenas um registro periférico, mas uma fonte de informação rica e complementar aos dados estruturais. O trabalho demonstra que a classificação dos comentários é viável e confiável, que essas classificações influenciam a duração dos bugs de forma significativa, e que a combinação entre características textuais e estruturais permite a

construção de modelos que capturam de maneira mais profunda o comportamento dos relatórios ao longo do seu ciclo de resolução. Esses achados reforçam a importância de incorporar análises linguísticas e históricas nas abordagens preditivas sobre manutenção de software, abrindo caminho para novas investigações que integrem comunicação, estrutura e tempo em análises automatizadas de qualidade e eficiência em ambientes de rastreamento de bugs.

9.2 Limitações do Trabalho

Apesar dos resultados consistentes obtidos, este estudo apresenta limitações que devem ser consideradas na interpretação dos achados. Primeiramente, a classificação manual dos comentários, embora cuidadosamente validada, foi realizada sobre um subconjunto do universo total de comentários disponíveis, o que pode introduzir vieses relacionados à distribuição das classes menos frequentes.

Adicionalmente, a rotulação manual foi conduzida por um único anotador (autor principal do estudo), não tendo sido realizada dupla anotação independente. Consequentemente, não foi possível calcular medidas formais de concordância interavaliadora, como o coeficiente Kappa de Cohen. Embora as decisões de classificação e os casos ambíguos tenham sido discutidos com os orientadores ao longo do processo, essa limitação pode introduzir vieses individuais na aplicação da taxonomia.

Outra limitação metodológica decorre da adoção de uma taxonomia com rótulo único para o conteúdo textual dos comentários. Embora essa decisão tenha favorecido a consistência do processo de anotação ao exigir que cada comentário recebesse um único rótulo predominante com base em critérios estáveis, reduzindo variações na aplicação da taxonomia em casos limítrofes e facilitando a padronização do dataset para treinamento e comparação dos modelos, ela simplifica casos em que um mesmo comentário apresenta simultaneamente múltiplas funções comunicativas, como discussão, questionamento e proposta de solução parcial. Assim, comentários semanticamente híbridos foram reduzidos à classe considerada predominante, o que pode diminuir a representatividade de casos com sobreposição discursiva.

Além disso, as análises de importância de atributos ficaram restritas a modelos baseados em árvores, como *Random Forest* e *Decision Tree*, uma vez que modelos lineares, de margens máximas e probabilísticos (como *Logistic Regression*, *SVM* e *Naive Bayes*) não oferecem mecanismos diretos e comparáveis de interpretação de atributos. Isso limita uma análise interpretativa mais ampla e homogênea entre todos os modelos avaliados.

No caso dos modelos não supervisionados e das LLMs, as limitações observadas estão fortemente relacionadas tanto às características dos algoritmos quanto à natureza altamente ambígua e sobreposta dos comentários de relatórios de bugs. Em especial, o uso de LLMs foi restrito a configurações exploratórias, sem ajuste fino específico para o domínio, o que pode ter impactado sua capacidade de seguir esquemas rígidos de classificação.

9.3 Implicações

Os resultados deste estudo apresentam implicações relevantes tanto para a pesquisa acadêmica em relatórios de bugs quanto para a prática profissional no desenvolvimento e manutenção de software.

9.3.1 Implicações para a pesquisa em Relatórios de Bugs

Do ponto de vista acadêmico, este estudo contribui ao demonstrar que a seção de comentários de relatórios de bugs pode ser analisada de forma sistemática e explorada como uma fonte rica de informações para compreensão do processo de resolução de defeitos. A taxonomia proposta e a classificação manual associada oferecem um conjunto de dados rotulados que pode ser utilizado como base para investigações futuras, permitindo a replicação e a extensão dos experimentos realizados.

Além disso, os resultados evidenciam que modelos supervisionados tradicionais, quando combinados com técnicas adequadas de balanceamento e pré-processamento textual, são eficazes para a classificação estruturada de comentários. Por outro lado, os experimentos com abordagens não supervisionadas e modelos de linguagem de grande escala utilizados via prompting demonstraram limitações significativas para tarefas com

taxonomias rígidas, reforçando a importância de controle metodológico em cenários de classificação formal.

9.3.2 Implicações para engenheiros de software

Para engenheiros de software, líderes técnicos e gestores de equipes de manutenção, os resultados indicam que a classificação automática de comentários e a previsão da duração dos relatórios podem apoiar decisões ainda nos estágios iniciais do ciclo de vida do bug, desde que baseadas nas informações já disponíveis naquele momento. O modelo não depende do encerramento completo do relatório, mas sim de sinais precoces, como as primeiras alterações de status, movimentações em campos administrativos e os comentários iniciais publicados. A identificação de padrões comunicativos, como discussões extensas, solicitações recorrentes de esclarecimento ou tentativas de solução parcial, pode atuar como indicador complementar da complexidade do problema. Embora os sinais estruturais tenham se mostrado mais determinantes na predição da duração, as classificações dos comentários agregam nuances relevantes, especialmente na distinção de casos intermediários ou limítrofes. Essa capacidade de estimativa antecipada é particularmente útil para gerentes de projeto e mantenedores de equipes, que podem utilizar os sinais produzidos pelo modelo para identificar relatórios com maior probabilidade de resolução prolongada, antecipar gargalos técnicos, redistribuir responsabilidades ou ajustar o planejamento de sprints e priorização do backlog.

9.3.3 Implicações para implementadores de ferramentas

Para implementadores e desenvolvedores de ferramentas de apoio ao desenvolvimento de software, os modelos e técnicas apresentados neste estudo podem servir como base para a criação de funcionalidades inteligentes em sistemas de rastreamento de bugs. A aplicação prática dessas técnicas pode ser incorporada diretamente a sistemas de rastreamento de bugs por meio de funcionalidades inteligentes que operem de forma incremental. Entre as possibilidades estão a classificação automática do tipo de

comentário no momento de sua publicação, a geração de indicadores de instabilidade baseados no histórico precoce de modificações e a emissão de alertas associados a relatórios com maior probabilidade de se tornarem casos de longa duração.

Diferentemente de abordagens retrospectivas, o modelo proposto pode ser aplicado continuamente à medida que novos eventos são registrados. Sempre que um comentário é adicionado ou uma modificação estrutural ocorre, a previsão pode ser recalculada com base no conjunto atualizado de informações iniciais. Isso permite transformar o sistema de rastreamento de bugs em um instrumento ativo de apoio à decisão, fornecendo sinais antecipados sobre risco de atraso e complexidade crescente.

Em contextos com alto volume de relatórios, essa abordagem pode contribuir para uma gestão mais proativa do fluxo de manutenção, reduzindo dependência exclusiva de análise manual e favorecendo decisões baseadas em evidências estruturais e comunicacionais já disponíveis no início do ciclo de vida do relatório.

9.4 Contribuições do Trabalho

Os resultados obtidos ao longo deste estudo confirmam as contribuições propostas no capítulo introdutório e evidenciam avanços relevantes para a pesquisa empírica em engenharia de software e para a análise de relatórios de bugs.

A taxonomia de classes desenvolvida mostrou-se adequada para representar os diferentes papéis comunicativos, técnicos e informativos presentes nos comentários de relatórios de bugs, permitindo uma caracterização mais precisa desses artefatos. A construção de um conjunto de dados anotado manualmente, baseado nessa taxonomia, possibilitou a condução de experimentos supervisionados confiáveis e forneceu uma base sólida para análises empíricas e investigações futuras.

Os experimentos realizados demonstraram a viabilidade da classificação automática de comentários por meio de algoritmos de aprendizagem de máquina, mesmo em cenários de forte desbalanceamento entre classes. Além disso, a análise empírica conduzida evidenciou relações consistentes entre padrões comunicativos observados nos comentários e o tempo de resolução dos relatórios de bugs.

Por fim, os modelos propostos para a previsão da duração de resolução dos relatórios, combinando informações iniciais dos relatórios, histórico de modificações e sinais derivados da classificação dos comentários, indicaram que aspectos comunicativos e estruturais exercem influência conjunta sobre o ciclo de vida dos relatórios de bugs, ampliando a compreensão sobre os fatores associados à sua resolução.

9.5 Trabalhos Futuros

Os resultados deste estudo abrem uma série de possibilidades para avanços metodológicos e científicos que podem fortalecer a compreensão sobre a dinâmica dos relatórios de bugs, bem como aprimorar as técnicas de análise utilizadas. Uma das direções mais promissoras envolve aprofundar a exploração de modelos não supervisionados para detecção de padrões comunicativos e de comportamento, especialmente considerando a natureza altamente heterogênea e ruidosa dos comentários em plataformas como o Bugzilla. Embora os experimentos realizados tenham demonstrado as limitações de métodos tradicionais como *K-Means*, *DBSCAN* e *HDBSCAN* para recuperar classes semanticamente coerentes, elas abrem espaço considerável para tentativas com abordagens mais modernas, como *BERTopic* [64], *Top2Vec* [65], modelos baseados em *embeddings* contextualizados (*Sentence-BERT* [66]) e técnicas de *clustering* hierárquico assistido por similaridade semântica. Outros métodos podem ser capazes de identificar estruturas discursivas que não se enquadram estritamente na classificação proposta, oferecendo visões mais amplas ou mais finas sobre o conteúdo dos comentários.

Uma direção futura particularmente relevante consiste em explorar abordagens de classificação *multilabel* ou com sobreposição explícita entre classes. Como muitos comentários de relatórios de bugs desempenham mais de uma função ao mesmo tempo, como por exemplo, combinando discussão, pergunta e tentativa de solução, modelos capazes de representar essa natureza híbrida podem produzir descrições mais fiéis da dinâmica comunicativa observada nos bug trackers. A investigação de esquemas de anotação com *overlapping* e de métodos de aprendizagem adequados a esse cenário

pode ampliar o poder explicativo da taxonomia e reduzir a perda de nuance, isto é, a redução de comentários híbridos a uma única categoria, o que pode ocultar funções simultâneas como questionamento, discussão e tentativa de solução presentes no mesmo texto associada à classificação exclusiva adotada neste trabalho.

Outra direção importante envolve o uso de modelos de larga escala (LLMs), que vêm evoluindo rapidamente e já demonstram capacidades sofisticadas em tarefas de classificação, sumarização, rotulação automática e geração de insights a partir de dados textuais complexos. Embora os resultados deste trabalho tenham mostrado dificuldades no uso de T5 e Gemini para classificação estrita, essa limitação decorreu principalmente da sensibilidade desses modelos à formatação rígida exigida e da ausência de "*instruction tuning"* específico para o domínio de relatórios de bugs. Trabalhos futuros podem explorar modelos mais recentes, treinados com capacidades de seguir instruções de forma robusta, ou até desenvolver pequenos modelos ajustados localmente especialmente para o estilo de comunicação encontrado em *bug trackers*. A utilização de métodos modernos, como LLMs com *instruction tuning* específico para engenharia de software, modelos híbridos supervisionados–não supervisionados e abordagens baseadas em aprendizado contrastivo, podem mitigar as inconsistências observadas nos experimentos e expandir o potencial dos LLMs para a classificação, detecção de anomalias, identificação de tom, extração de entidades, sumarização contextualizada e análise das relações entre desenvolvedores.

Referências

[1] PRESSMAN, R. S.; MAXIM, B. R. *Software engineering: a practitioner's approach*. 8. ed. New York: McGraw-Hill Education, 2014.

[2] LAUDON, K. C.; LAUDON, J. P. *Management information systems: managing the digital firm*. 15. ed. New York: Pearson, 2018.

[3] GROMOVA, A.; ITKIN, I.; PAVLOV, S.; KOROVAYEV, A. Raising the quality of bug reports by predicting software defect indicators. In: *IEEE INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, RELIABILITY AND SECURITY COMPANION (QRS-C)*, 2019. p. 198–204. DOI: https://doi.org/10.1109/QRS-C.2019.00048. Acesso em: 9 jan. 2026.

[4] RAYMOND, E. S. *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. Sebastopol: O'Reilly Media, 2001.

[5] BREU, S.; PREMRAJ, R.; SILLITO, J.; ZIMMERMANN, T. *Frequently asked questions in bug reports*. Technical Report 2009-924-03. Calgary: University of Calgary, 2009. Disponível em: https://www.researchgate.net/publication/228427851. Acesso em: 30 dez. 2025.

[6] CATOLINO, G.; PALOMBA, F.; ZAIDMAN, A.; FERRUCCI, F. Not all bugs are the same: understanding, characterizing, and classifying bug types. *Journal of Systems and Software*, v. 152, p. 165–181, 2019. DOI: https://doi.org/10.1016/j.jss.2019.03.002. Acesso em: 9 jan. 2026.

[7] BUGZILLA. *Bugzilla*. Disponível em: https://www.bugzilla.org/. Acesso em: 5 maio 2024.

[8] MOZILLA FOUNDATION. *Fundação Mozilla*. Disponível em: https://foundation.mozilla.org/pt-BR/. Acesso em: 5 maio 2024.

[9] ECLIPSE FOUNDATION. *Fundação Eclipse*. Disponível em: https://www.eclipse.org/org/foundation/. Acesso em: 5 maio 2024.

[10] MITCHELL, T. M. *Machine learning*. New York: McGraw-Hill, 1997.

[11] LIMSETTHO, N.; HATA, H.; MONDEN, A.; MATSUMOTO, K. Automatic unsupervised bug report categorization. In: *INTERNATIONAL WORKSHOP ON EMPIRICAL SOFTWARE ENGINEERING IN PRACTICE (IWESEP)*, 6., 2014, Osaka. p. 7–12. DOI: https://doi.org/10.1109/IWESEP.2014.8. Acesso em: 30 dez. 2025.

[12] MIN, B. et al. Recent advances in natural language processing via large pre-trained language models: a survey. *ACM Computing Surveys*, v. 56, n. 2, 2024. DOI: https://doi.org/10.1145/3605943.

[13] BUGZILLA. *Exemplo de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=1655944. Acesso em: 8 jun. 2024.

[14] CHAWLA, N. V. et al. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 2002.

[15] BUGZILLA. *Enter a bug*. Disponível em: https://bugzilla.mozilla.org/enter_bug.cgi. Acesso em: 8 jun. 2024.

[16] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=955888. Acesso em: 19 maio 2024.

[17] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=955913. Acesso em: 14 jun. 2024.

[18] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=955957. Acesso em: 14 jun. 2024.

[19] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=956044. Acesso em: 22 jun. 2024.

[20] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=956066. Acesso em: 22 jun. 2024.

[21] KARIM, M. R.; IHARA, A.; CHOI, E.; IIDA, H. Identifying and predicting key features to support bug reporting. *Software Maintenance and Evolution*, v. 31, 2019. DOI: https://doi.org/10.1002/smr.2184. Acesso em: 9 jan. 2026.

[22] BUGZILLA. *Comentário de relatório de bug*. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=955934. Acesso em: 22 jun. 2024.

[23] CALIXTO, F.; RAMALHO, F.; MASSONI, T.; FERREIRA, J. Investigating bug report changes in Bugzilla. In: *INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS (ICEIS)*, 25., 2023. Anais […]. p. 55–64. DOI: https://doi.org/10.5220/0011847600003467. Acesso em: 9 jan. 2026.

[24] FERREIRA, J. M. S. *Relating bug report fields with resolution status: a case study with Bugzilla*. 2023. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Campina Grande, 2023.

[25] SASSO, T. D.; MOCCI, A.; LANZA, M. What makes a satisficing bug report? In: *IEEE INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, RELIABILITY AND SECURITY (QRS)*, 2016. p. 164–174. DOI: https://doi.org/10.1109/QRS.2016.28. Acesso em: 9 dez. 2025.

[26] BÖCK, H. et al. *The definitive guide to NetBeans platform*. Berkeley: Apress, 2009.

[27] POWERS, D. M. W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, v. 2, n. 1, p. 37–63, 2011.

[28] VAN RIJSBERGEN, C. J. *Information retrieval*. 2. ed. London: Butterworths, 1979.

[29] ROUSSEEUW, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, v. 20, p. 53–65, 1987. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. Acesso em: 9 dez. 2025.

[30] JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. *ACM Computing Surveys*, v. 31, n. 3, p. 264–323, 1999. DOI: https://doi.org/10.1145/331499.331504. Acesso em: 9 dez. 2025.

[31] HUBERT, L.; ARABIE, P. Comparing partitions. *Journal of Classification*, v. 2, p. 193–218, 1985. DOI: https://doi.org/10.1007/BF01908075. Acesso em: 9 dez. 2025.

[32] XIA, X. et al. Automated bug report field reassignment and refinement prediction. *IEEE Transactions on Reliability*, v. 65, n. 3, p. 1094–1113, 2016. DOI: https://doi.org/10.1109/TR.2015.2484074. Acesso em: 9 de jan. 2026.

[33] ZHANG, F. et al. An empirical study on factors impacting bug fixing time. In: *WORKING CONFERENCE ON REVERSE ENGINEERING (WCRE)*, 2012. Anais […]. DOI: https://doi.org/10.1109/WCRE.2012.32. Acesso em: 18 dez. 2025.

[34] SCRIPTS. *Colab notebooks*. Disponível em: https://drive.google.com/drive/folders/1LY_GLPHe5mdMu1VsMzI7WF3ujJe_zYA8. Acesso em: 9 jan. 2026.

[35] HARRIS, Z. S. Distributional structure. *Word*, v. 10, n. 2–3, p. 146–162, 1954.

[36] BISONG, E. *Building machine learning and deep learning models on Google Cloud Platform*. Berkeley: Apress, 2019. DOI: https://doi.org/10.1007/978-1-4842-4470-8. Acesso em: 9 jan. 2026.

[37] PEDREGOSA, F. et al. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

[38] MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. *Introduction to information retrieval*. Cambridge: Cambridge University Press, 2008.

[39] CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, v. 20, p. 273–297, 1995. DOI: https://doi.org/10.1007/BF00994018. Acesso em: 9 jan. 2026.

[40] BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001. DOI: https://doi.org/10.1023/A:1010933404324. Acesso: 12 jan. 2026

[41] HOSMER, D. W.; LEMESHOW, S. *Applied logistic regression*. 2. ed. New York: Wiley, 2000.

[42] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In: *BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY*, 5., 1967. p. 281–297.

[43] ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases. In: *KDD*, 1996.

[44] ABADI, M. et al. TensorFlow: a system for large-scale machine learning. In: *OSDI*, 2016.

[45] CHOLLET, F. *Deep learning with Python*. New York: Manning, 2017.

[46] LE CUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–444, 2015. DOI: https://doi.org/10.1038/nature14539. Acesso em: 9 jan. 2026.

[47] CAMPELLO, R. J. G. B.; MOULAVI, D.; SANDER, J. Density-based clustering based on hierarchical density estimates. In: *PAKDD*, 2013. DOI: https://doi.org/10.1007/978-3-642-37456-2_14. Acesso em: 9 jan. 2026.

[48] MCKINNEY, W. Data structures for statistical computing in Python. In: *PYTHON IN SCIENCE CONFERENCE*, 2010.

[49] HARRIS, C. R. et al. Array programming with NumPy. *Nature*, v. 585, p. 357–362, 2020. DOI: https://doi.org/10.1038/s41586-020-2649-2. Acesso em: 9 jan. 2026.

[50] SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, v. 24, n. 5, p. 513–523, 1988. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. Acesso em: 9 jan. 2026.

[51] QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, p. 81–106, 1986. DOI: https://doi.org/10.1007/BF00116251. Acesso em: 21 dez. 2025.

[52] GEMINI TEAM. Gemini: a family of highly capable multimodal models. *arXiv*, 2023. DOI: https://doi.org/10.48550/arXiv.2312.11805. Acesso em: 22 dez. 2025.

[53] RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, v. 21, 2020.

[54] HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, p. 1263–1284, 2009. DOI: https://doi.org/10.1109/TKDE.2008.239. Acesso em: 9 jan. 2026.

[55] KIKAS, R.; DUMAS, M.; PFAHL, D. Using dynamic and contextual features to predict issue lifetime in GitHub projects. In: *MINING SOFTWARE REPOSITORIES (MSR)*, 2016. DOI: https://doi.org/10.1145/2901739.2901751. Acesso em: 22 dez. 2025.

[56] BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, p. 281–305, 2012.

[57] WILLMOTT, C. J.; MATSUURA, K. Advantages of the mean absolute error over the root mean square error. *Climate Research*, v. 30, p. 79–82, 2005. DOI: https://doi.org/10.3354/cr030079. Acesso em: 13 jan. 2026

[58] HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of statistical learning*. 2. ed. New York: Springer, 2009.

[59] SOKAL, R. R.; ROHLF, F. J. *Biometry*. 3. ed. New York: Freeman, 1995.

[60] CHEN, T.; GUESTRIN, C. XGBoost: a scalable tree boosting system. In: *KDD*, 2016. DOI: https://doi.org/10.1145/2939672.2939785. Acesso: 9 de jan. 2026.

[61] KE, G. et al. LightGBM: a highly efficient gradient boosting decision tree. In: *NEURIPS*, 2017. DOI: https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html . Acesso em: 9 de jan. 2026.

[62] LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: *NEURIPS*, 2017. DOI:

https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html. Acesso em: 26 dez. 2025.

[63] RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. Why should I trust you? Explaining the predictions of any classifier. In: *KDD*, 2016. DOI: https://doi.org/10.1145/2939672.2939778. Acesso em: 9 jan. 2026.

[64] GROOTENDORST, M. BERTopic: neural topic modeling with a class-based TF-IDF procedure. *arXiv*, 2022. Disponível em: https://arxiv.org/abs/2203.05794. Acesso em: 12 jan. 2026.

[65] ANGELI, D.; CARDOSO, J. Top2Vec: distributed representations of topics. *arXiv*, 2020. Disponível em: https://arxiv.org/abs/2008.09470. Acesso em: 12 jan. 2026.

[66] REIMERS, N.; GUREVYCH, I. Sentence-BERT: sentence embeddings using Siamese BERT-networks. In: *EMNLP*, 2019. DOI: https://doi.org/10.18653/v1/D19-1410. Acesso em: 9 jan. 2026.

[67] PEFFERS, K.; TUUNANEN, T.; ROTHENBERGER, M. A.; CHATTERJEE, S. A design science research methodology for information systems research. *Journal of Management Information Systems*, v. 24, n. 3, p. 45–77, 2007. DOI: https://doi.org/10.2753/MIS0742-1222240302. Acesso em: 13 jan. 2026.