MICHELLEN DE CARVALHO SILVA

UMA ANÁLISE E CLASSIFICAÇÃO AUTOMÁTICA DE *BUGS*

LAVRAS-MG

2021

MICHELLEN DE CARVALHO SILVA

UMA ANÁLISE E CLASSIFICAÇÃO AUTOMÁTICA DE *BUGS*

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para obtenção do título de Bacharel.

Prof. Dr. Antônio Maria P. de Resende
Orientador

LAVRAS-MG
2021

MICHELLEN DE CARVALHO SILVA

UMA ANÁLISE E CLASSIFICAÇÃO AUTOMÁTICA DE *BUGS*
AUTOMATIC *BUGS* ANALYSIS AND CLASSIFICATION

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para obtenção do título de Bacharel.

APROVADA em 14 de abril de 2021.
Dr. Antônio Maria P. de Resende UFLA
Dra. Renata Teles Moreira UFLA
Me. Douglas Nunes de Oliveira UFLA

Prof. Dr. Antônio Maria P. de Resende
Orientador

LAVRAS-MG
2021

RESUMO

Os repositórios disponíveis na web tem contribuído bastante com a resolução de issues, conhecidos como *bugzilla*, são projetos de software livre que permite o gerenciamento das mesmas. Entretanto, a classificação manual dessas issues é lenta e cansativa, devido às mudanças na equipe de desenvolvimento ou até mesmo na procura de um desenvolvedor com experiência similar a issue reportada. Com isso, a classificação automática é bastante vantajosa, pois desta forma o bug será encaminhado para o desenvolvedor experiente na área em que o mesmo foi classificado, diminuindo o tempo de correção, o custo da triagem dos erros e o retrabalho do desenvolvedor de encaminhar para os especialistas adequados. O objetivo deste trabalho é implementar um protótipo que realiza a classificação automática dos *bugs* do *bugzilla* do Gentoo. Tal protótipo foi realizado através da escolha do repositório, um pré-processamento na base de dados e a mineração dos mesmos. Os classificadores utilizados foram o *Decision Tree* e *Random Forest*, trabalhados em uma base de 1000 *bugs* e o que apresentou melhor resultado foi o *Random Forest* com a quantidade de elementos maiores.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A manutenção de um sistema pode gerar erros em suas funcionalidades, sendo importante a realização de um controle de qualidade no qual permite reduzir essa probabilidade de erros e custos de manutenção. Para isso são necessárias ajudas tecnológicas responsáveis por realizar testes constantes, análise de códigos e equipes para descobrir *bugs* nos sistemas e relatá-los em forma de *issue*. Muitos *bugs* são registrados em projetos livres como Eclipse, LibreOffice, Mozilla, Gentoo, entre outros. Um *bug* é identificado quando um serviço prestado pelo software se desvia do serviço correto que ele deveria prestar (LYU, 1996).

O Bugzilla é uma ferramenta que permite o gerenciamento de issues de software de projetos como o Gentoo, contendo informações como registro de defeitos (*bugs*), sugestões de correções ou aprimoramentos a serem tratados pelos colaboradores do projeto, desde quando o usuário reporta o bug até o momento em que ele é resolvido. É um projeto de software livre, no qual concede a liberdade aos próprios usuários de identificarem o problema e relatar no bugzilla. Assim, outros usuários irão analisar e confirmar a presença da falha descrita, propondo várias soluções até a resolução do problema. Cada nova *issue* reportada deve ser identificada como um bug ou solicitação de aprimoramento. Além disto, o administrador de issues tem a responsabilidade de atribui-la a um colaborador assim que a mesma é cadastrada, adotando como critério de distribuição sua percepção inicial do tipo de issue.

No entanto, esse processo manual de atribuição de issue a um colaborador apropriado de acordo com suas características é lento e cansativo, devido às mudanças na equipe de desenvolvimento ou até mesmo encontrar um desenvolvedor com experiência similar a issue reportada. Segundo (JEONG, 2009) 44% dos *bugs* reportados foram atribuídos ao desenvolvedor errado, talvez por acidente ou por falta de experiência do mesmo, aumentando consideravelmente o tempo de correção de um bug.

Os *bugs* segundo a ISO/IEC25010, poderiam ser classificados em características de qualidade, sendo elas: Adequação Funcional, Eficiência de Desempenho, Compatibilidade, Usabilidade, Portabilidade, Confiabilidade, Segurança e Manutenção

A classificação dos *bugs* é de extrema importância, pois desta forma o bug será encaminhado para o desenvolvedor experiente na área em que o mesmo foi classificado, diminuindo o tempo de correção, o custo da triagem dos erros e o retrabalho do desenvolvedor de encaminhar para os especialistas adequados.

Considerando a contextualização exposta acima, o objetivo deste trabalho é desenvolver um modelo de classificação automática dos *Bugs* encontrados no bugzilla do Gentoo e implementar um protótipo, no qual classificará os bugs como de segurança ou não.

Este trabalho está estruturado da seguinte forma. Na seção 2 apresenta-se o referencial teórico, no qual serão mostrados os conceitos sobre a Norma ISO/IEC 25010, Mineração de Dados e *Machine learning*, a Metodologia está expressa na seção 3, sendo dividida em 5 etapas, sendo elas: i) Seleção do Repositório; ii) Seleção dos Dados; iii) Mineração dos Dados; iv) *Machine learning*; e v) Resultados. O Desenvolvimento está apresentado na seção 4, no qual detalha todas as etapas que constituiu a metodologia. A Conclusão está exposta na seção 5, apresentando as considerações finais do trabalho, os melhores resultados e as possíveis melhorias para trabalhos futuros.

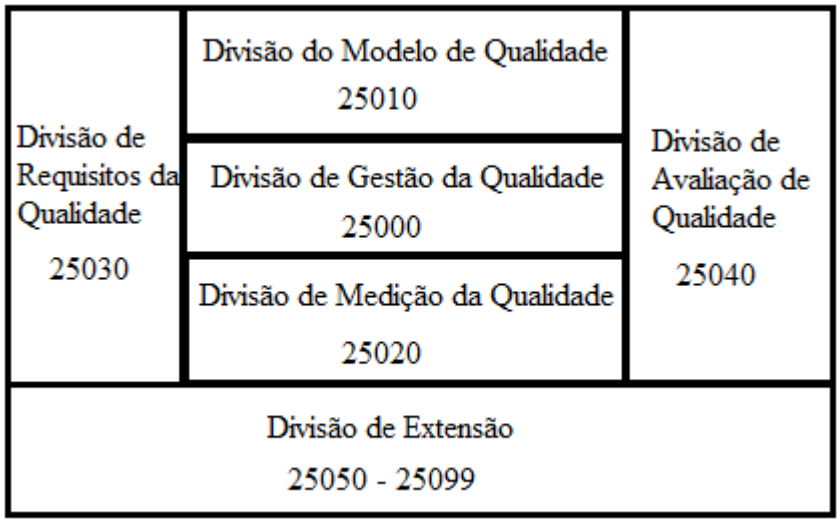
2. REFERENCIAL TEÓRICO

Para o desenvolvimento de uma aplicação que classifica os *bugs* do Gentoo, cadastrados no Bugzilla, como bug de segurança e bug de não segurança, será necessário abordar assuntos como ISO/IEC 25010, *bugs*, *Machine learning*, pré-processamento e mineração de dados.

2.1. Norma ISO/IEC 25010

A norma ISO/IEC 25010 faz parte da família ISO/IEC 25000, considerada uma família de normas organizadas para avaliar qualidade de software e especificação de requisitos, utilizando processo de medição. Essa família da ISO/IEC 25000, também conhecida como SQuaRE, foi definida dentro de um intervalo numérico de 25000 à 25099, contemplando as várias etapas do ciclo de vida de um software (*STANDARD*, 2010). A numeração final, ilustrada na Figura 1, foi aprovada e aplicada em maio de 2002:

Figura 1 - Arquitetura atual da ISO/IEC 25000



Fonte: Adaptação - ISO 25000 *Standards* (2020)

Segundo (STANDARD, 2010), a ISO/IEC 25010 representa a divisão de modelo de qualidade para software (Product Quality Model), auxiliando na avaliação de desenvolvedores, gerentes, compradores e usuários. Esse modelo é composto por 8 características subdividas, relacionadas às propriedades estáticas do software e propriedades dinâmicas do sistema, sendo elas:

- Funcionalidade: Completude, corretude e adequação funcional;
- Eficiência de Desempenho: Comportamento em relação ao tempo, aos recursos e capacidade;
- Compatibilidade: Coexistência e interoperabilidade;

- Usabilidade: Conhecimento adequado, apreensibilidade, operacionalidade, acessibilidade, proteção de erros de usuário e estética da interface com o usuário;
- Confiabilidade: Maturidade, tolerância a falhas, recuperabilidade e disponibilidade;
- Segurança: Confidencialidade, integridade, não-repúdio, responsabilidade e autenticidade;
- Manutenibilidade: Analisabilidade, modificabilidade, modularidade, testabilidade e reusabilidade;
- Portabilidade: Adaptabilidade, instalabilidade e substituibilidade.

Os *bugs* analisados neste trabalho poderiam ser classificados em qualquer uma das características acima.

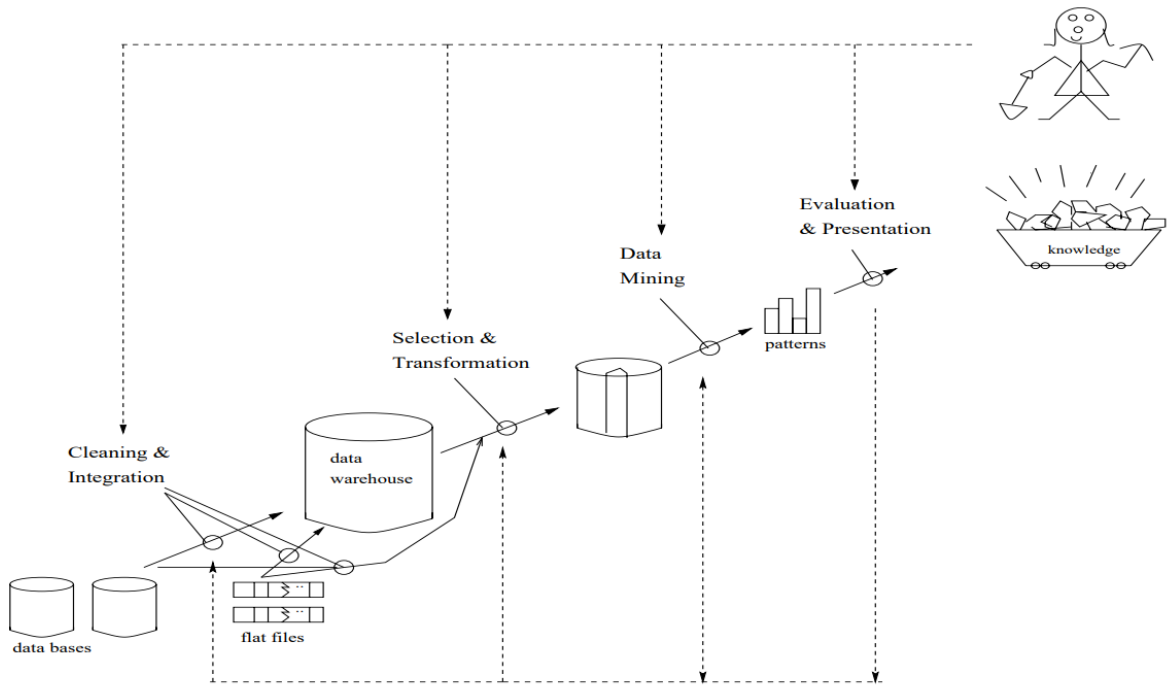
2.2. Mineração de Dados

Segundo (AMO, 2003), a Mineração de Dados relacionada a banco de dados, trata do processo de extração ou mineração de conhecimento a partir de grandes volumes de dados. Fayyad, Piatetsky-Shapiro e Smyth (1996) considera mineração de dados como um processo de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis disponíveis nos dados.

De acordo com (Han, Kamber e Pei, 2012), a mineração de dados refere-se a extrair ou minerar conhecimento de grandes quantidades de dados. Também conhecida simplesmente como uma etapa essencial no processo de conhecimento descoberta em bancos de dados. A descoberta de conhecimento como um processo é apresentada na Figura 2, com as seguintes etapas:

- Limpeza de dados: Remove ruído ou dados irrelevantes;
- Integração de dados: Pode ocorrer combinação de várias fontes de dados;
- Seleção de dados: Seleção dos dados relevantes para análise do banco de dados;
- Transformação de dados: Transformação ou consolidação dos dados de formas apropriadas para a mineração;
- Mineração dos dados: Aplicação de métodos inteligentes para extrair padrões de dados;
- Avaliação de padrões: Identificação dos padrões que representam o conhecimento com base nas medidas de interesse.
- Apresentação do conhecimento: Utilização das técnicas de visualização e representação de conhecimento para apresentar os resultados extraídos ao usuário.

Figura 2 - Mineração de Dados



Fonte: Livro - *Data Mining: Concepts and Techniques*

A Mineração de Dados é realizada em qualquer formato do dado, com isso, temos a Mineração de Textos, no qual aplica as mesmas funções analíticas da Mineração de Dados (GOMES, 2013), porém para dados textuais.

Segundo (HEARST, 1999), os dados textuais possuem uma fonte de informação bastante rica, e um formato difícil de extrair de maneira automatizada, devido ao fato de ser um grande volume de texto em linguagem natural não estruturada. Esse grande conjunto de informação textual não estruturada não pode ser diretamente utilizada por computadores para extração de conhecimento, já que os mesmos a tratam apenas como uma sequência de caracteres. Por conseguinte, é necessário diferentes técnicas para facilitar a Mineração de Dados. Um exemplo a ser citado é o pré-processamento, em que auxilia a extração de conhecimento dos respectivos dados.

De modo geral os dados armazenados em bases de dados não estão em formato adequado para extração de conhecimento, assim sendo faz-se necessária a aplicação de pré-processamento antes da etapa de mineração.

O pré-processamento auxilia na coleta, organização e tratamento de dados e tem o propósito de prepará-los para a mineração de dados. Segundo (TAN; STEINBACH; KUMAR,

2009), os dados podem ser coletados e armazenados de diversas maneiras, em razão disso, essa fase é considerada a mais trabalhosa do processo.

Conforme (CORRÊA, 2007), o pré-processamento possui quatro etapas: Seleção de dados; limpeza de dados; codificação dos dados; enriquecimento dos dados.

A etapa de seleção de dados envolve a definição e especificação da base de dados a ser utilizada. Nesta etapa ocorre a escolha do conjunto de dados que será utilizada para análise.

Logo após, tem-se a etapa de limpeza de dados, a qual se faz necessária pois, segundo NAVEGA (2002), as bases de dados são dinâmicas, incompletas, redundantes, ruidosas e esparsas, necessitando de um pré-processamento para limpá-las. Na etapa de limpeza de dados, de acordo com GOLDSCHMIDT & PASSOS (2005) as seguintes funções que podem ser aplicadas para a limpeza de dados são:

• Limpeza de informações ausentes: compreende a eliminação de valores ausentes em conjunto de dados;

• Limpeza de inconsistências: abrange a identificação e a eliminação de valores inconsistentes em conjunto de dados;

• Limpeza de valores não pertencentes ao domínio: compreende a identificação e a eliminação de valores que não pertençam ao domínio dos atributos do problema

Na etapa de codificação, os dados podem ser transformados alterando o formato da base de dados e evoluindo suas informações. E por fim, o enriquecimento dos dados, no qual com a junção de todas essas informações, realiza a escolha do algoritmo para a mineração de dados.

Após o pré-processamento, os dados estão preparados para a mineração de dados propriamente dita. Essa etapa é crucial para garantir a eficiência do resultado do projeto e depende diretamente da escolha de técnicas e suas combinações.

As técnicas de Mineração de Dados utilizadas no presente trabalho serão a seleção, pré-processamento, transformação, mineração e interpretação. No pré-processamento serão viáveis todas as etapas conforme (CORRÊA, 2007) nos propôs: Seleção de Dados, Limpeza de Texto, Codificação e Enriquecimento do texto.

Algumas técnicas, funções utilizadas na etapa de mineração de dados no presente trabalho estão sendo apresentadas abaixo:

- Part-of-speech tagging (POS-Tagging) trata-se da identificação da classe gramatical dos tokens de um texto. Como uma parte importante do processo de NLP, POS-Tagging é um campo muito estudado da área, com ferramentas do estado da arte sendo capazes de

alcançar mais de 97% de acurácia na classificação de tokens (DERCZYNSKI et al., 2013).

- Lematização, segundo (De Lucca e Nunes, 2002), consiste em reunir todas as ocorrências da mesma palavra sob uma única forma, o lema, como acontece num dicionário, em vez de apresentá-las tal como aparecem nos textos, com variações no gênero, no número ou na grafia. Este método causa uma menor variação de palavras, tornando possível um maior número de conexões entre sentenças.
- Tokenização é o processo de dividir blocos de textos em frases, palavras, símbolos ou elementos significativos chamados de tokens (Verma et al. 2014). Seu principal objetivo é explorar palavras dentro de contextos frasais para identificação de palavras-chaves.
- *TF-IDF* (*Term Frequency-Inverse Document Frequency*) é uma técnica estatística que tem como principal utilidade descobrir palavras de importância em um texto não estruturado ou semi estruturado (LIMA e CASTRO, 2012). Atribui-se um valor a cada termo, que considera sua frequência no texto e em todos dos documentos da base, indicando sua importância. Na equação representada pela figura X: o *TF*, representado por TF (i, j), é o número de vezes que o termo i aparece no documento j. O *IDF*, representado como IDF (i, k), é o logaritmo do total de documentos (k) dividido pelo número de documentos que contém o termo i.

Figura 3 - Equação da técnica *TF-IDF*

$$TF - IDF_{(i,j,k)} = TF_{(i,j)} \times IDF_{(i,k)}$$

Fonte: Kido e Junior. (2013)

Essa técnica consiste em filtrar todo o texto e classificar as palavras com valores que representam um grau de afinidade com o texto (RAMOS, 2003). Termos que são comuns em um único ou pequeno grupo de documentos tendem a ter um elevado *TF-IDF* em relação a palavras que se repetem várias vezes, como artigos, pronomes e preposições.

- Segundo (Broder, Glassman, Manasse e Zweig, 1997) no âmbito computacional, um n-grama é uma sequência imediata de n itens de um determinado conjunto de dados. Os termos podem ser fonemas, sílabas, letras, palavras ou pares de bases de acordo com a aplicação. Os n -gramas normalmente são coletados de texto. Com isso, um n -grama

de tamanho 1 é referido como um "unigrama"; o tamanho 2 é um " bigrama " (ou um "digrama"); o tamanho 3 é uma “trigrama ". No presente trabalho utilizaremos unigrama e bigrama.

2.3. Machine learning

Sistemas de *machine learning* ou aprendizado de máquina são sistemas que aprendem a partir dos dados e que pretende tomar decisão com o mínimo de intervenção humana.

A *machine learning* tem uma hierarquia do aprendizado de sistema, que seria o processo de indução. A indução é o recurso mais utilizado pelo cérebro humano para derivar conhecimento novo. (MONARD; BARANAUSKAS, 2003) Para algum conceito ser aprendido na indução, gera-se várias hipóteses de conhecimento nos exemplos analisados, e essas hipóteses geradas podem ou não ser algo verdadeiro. (CARVALHO et al., 2011) O processo de indução é algo que se aplicado com o número baixo de dados, ou se as amostras não forem bem escolhidas e também as variáveis/atributos dos dados, as hipóteses obtidas podem ser de pouco valor. Por isso, o aprendizado de máquina precisa ter grandes volumes de amostras dados para conseguir aprender e retirar algo relevante para o problema que está sendo resolvido. E as variáveis nesse conjunto de dados, tragam valor e gerem o maior número de hipóteses para o algoritmo aprender. (CARVALHO et al., 2011)

A Figura 4 mostra a hierarquia de aprendizado, no topo o processo de aprendizado indutivo, em seguida para o supervisionado que seriam as tarefas preditivas (classificação e regressão) e os não supervisionados que seriam as descritivas (agrupamento, associação, sumarização), esses são os mais conhecidos e mais utilizados. (CARVALHO et al., 2011)

Figura 4 - Hierarquia *Machine Learning*

Fonte: Carvalho et al. (2011)

No presente trabalho será utilizado algoritmo supervisionado com a classificação.

O algoritmo preditivo ou supervisionado é uma função que, a partir de conjuntos de dados já rotulados e conhecidos, constrói um modelo que consegue estimar rótulos de novos dados. O rótulo é classe do dado ou atributo de saída. Se o domínio deste conjunto de dados com valores nominais é considerado um classificador e caso seja um conjunto infinito e ordenado de valores, tem um problema de regressão. Sendo através de um dado exemplo sem rótulo, consegue identificar umas das possíveis classes ou valor real para dado. (CARVALHO et al., 2011).

A categorização de texto é uma área de aprendizado de máquina relacionada à triagem de erros. A mesma envolve a classificação de texto de acordo com um conjunto de categorias e tem se tornado importante na área de mineração de texto. (SEBASTIAN, 2002).

Nos anos 80, os classificadores automáticos foram construídos de forma manual através de técnicas de engenharia do conhecimento, no qual era definida as regras lógicas por categorias ou tipos. Nos últimos anos surgiram outras técnicas de aprendizado para categorização de texto de forma automática, diminuindo o processo manual.

A árvore de decisão é uma representação de um classificador, onde é induzida a partir de um conjunto de treino das classes previamente conhecidas. (MICHEL, 1997).

(HASTIE, 2009) afirma que a estratégia utilizada pela árvore de decisão é de dividir para conquistar. Um problema é decomposto em vários subproblemas e a mesma estratégia é aplicada para cada subproblema gerado. Com isso, traz a desvantagem de instabilidade provocada por ruídos nos dados.

Segundo (Han, Kamber e Pei, 2012), as árvores de decisão funcionam como um fluxograma, sendo composto por nós internos, arestas e nós folhas. No momento em que uma instância necessita ser classificada, o caminho da árvore é percorrido.

O processo de classificação em uma árvore de decisão, acontece de maneira recursiva, conforme Figura 5, de modo que o nó inicial representa o conjunto de dados, em seguida deve ser avaliado se os objetos são da mesma classe, sendo esse o caso o nó é considerado um nó folha, caso contrário um atributo precisa ser usado para dividir os dados. Este processo deve ser executado recursivamente, ele pode ser descontinuado caso faltarem atributos para realizar testes de divisão ou caso todos os registros forem da mesma classe (CASTRO; FERRARI, 2016).

Figura 5 - Modelo baseado em árvore de decisão

Fonte: Castro e Ferrari (2016)

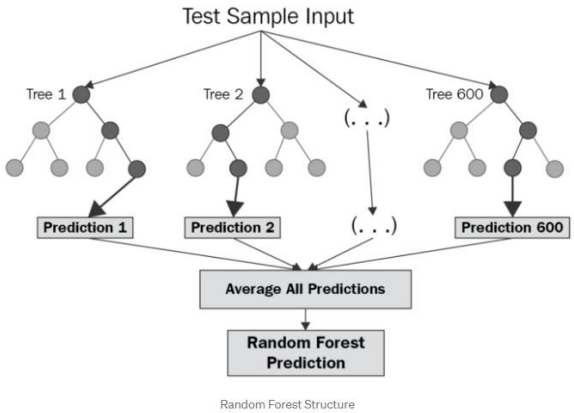
Dentre as técnicas de árvore de decisão, a *Random Forest* melhora a estabilidade e a precisão da árvore de decisão por incorporar um grande número de árvores em um único classificador (DINIZ,2003).

Random Forest são florestas aleatórias, um grupo de árvores de decisão, nos quais juntos formam uma floresta. Estas árvores são geradas com base em um atributo aleatório que é o responsável pela divisão em cada nó da árvore. A precisão de uma floresta aleatória é determinada de acordo com a força de cada classificador da árvore, e também o nível de dependência entre eles, o melhor modo de atingir essa precisão é mantendo a força dos classificadores e não aumentar a correlação entre eles (HAN; KAMBER e PEI, 2012).

(BREIMAN, 2001a) argumenta que as florestas aleatórias desfrutam de excepcional precisão de predição, e que essa precisão é obtida para uma ampla gama de configurações do único parâmetro de ajuste empregado.

Segundo (Chakure, 2019) floresta aleatória é um algoritmo de Aprendizagem Supervisionada que usa o método de aprendizagem de conjunto para classificação e regressão.

Figura 6 - Exemplo de Floresta Aleatória



Fonte: Chakure (2019)

A floresta aleatória é um comitê e não uma técnica de reforço. As árvores em florestas aleatórias são executadas em paralelo. Não há interação entre essas árvores durante a construção das árvores. Ele opera construindo uma infinidade de árvores de decisão no momento do treinamento e gerando a classe que é o modo das classes (classificação).

Uma floresta aleatória é um metaestimador (ou seja, combina o resultado de várias previsões) que agrega muitas árvores de decisão, com algumas modificações úteis:

- O número de recursos que podem ser divididos em cada nó é limitado a alguma porcentagem do total (conhecido como hiperparâmetro). Isso garante que o modelo de conjunto não dependa demais de qualquer recurso individual e faça uso justo de todos os recursos potencialmente preditivos.
- Cada árvore extrai uma amostra aleatória do conjunto de dados original ao gerar suas divisões, adicionando mais um elemento de aleatoriedade que evita o sobreajuste.

As modificações acima ajudam a evitar que as árvores sejam altamente correlacionadas.

Outros métodos para classificação de texto em linguagem natural usados atualmente são: Naive Bayes e K-nearest Neighbors (KNN).

Segundo (Han, Kamber e Pei, 2012), para avaliar o desempenho de um algoritmo de classificação, pode se avaliar sua capacidade preditiva. Conforme (Tan, Steinbach e Kumar, 2009), para a classificação de problemas binários, predicação entre duas classes, é utilizada a técnica na qual indica os dados com as classes de previsão e as classes atuais dos dados.

Mediante isso, os autores apresentam os termos utilizados:

• Verdadeiro Positivo (TP): número de exemplos positivos classificados corretamente;

• Falso Negativo (FN): número de exemplos negativos classificados incorretamente;

• Falso Positivo (FP): número de exemplos positivos classificados incorretamente;

• Verdadeiro Negativo (TN): número de exemplos negativos classificados corretamente.

Existem diversas métricas que podem ser utilizadas para a avaliação do desempenho de classificadores, sendo algumas apresentadas a seguir (*Scikit-Learn*):

• Revocação: taxa de objetos positivos verdadeiros classificados de forma correta; nenhum exemplo positivo é deixado de fora. Apresenta uma indicação do quanto do total de informação relevante foi recuperada.

Figura 7 - Equação Revocação

$$Revocação = \frac{VP}{VP + FN}$$

Fonte: *Scikit-Learn*

• Precisão: Taxa de objetos positivos classificados corretamente, nenhum exemplo negativo é incluído;

Figura 8 - Equação Precisão

$$Precisão = \frac{VP}{VP + FP}$$

Fonte: *Scikit-Learn*

• Medida F: Média harmônica entre as medidas de precisão e revocação, também conhecida como medida F1, pode ser definida como uma medida que busca relacionar as métricas de precisão e revocação a fim de obter uma medida de qualidade que equilibre a importância relativa destas duas métricas.

Figura 9 - Equação F1

$$F1 = 2\frac{P \times R}{P + R}$$

Fonte: *Scikit-Learn*

Para avaliação de desempenho tem-se o método de avaliação validação cruzada baseado no método de avaliação *holdout*.

O método *holdout* (THEODORIDIS e RIBERIO, 2003) é um tipo bem simples de método para teste de classificadores. Nele, a base de dados é dividida em dois conjuntos: conjunto de treinamento e conjunto de teste. O conjunto de treinamento fornece os dados para o treinamento da técnica utilizada e o conjunto de teste fornece dados novos, para testar a generalização da técnica. Devem ser feitos vários testes, acumulando-se os erros obtidos. É calculada, então, uma média desses erros para se avaliar o resultado. Uma vantagem desse

método é que ele é simples e computacionalmente rápido, porém sua avaliação pode ter uma alta variância.

Já o método de validação cruzada, os dados são divididos em uma parte para treinamento e outra parte para teste, porém se diferencia do *holdout* por continuar o treinamento para todos os padrões, ou seja, os conjuntos de treinamento e teste variam em um mesmo conjunto total de padrões. Um exemplo de validação cruzada (DELEN e DUDA, 2004) consiste em dividir o conjunto total de padrões em k grupos de tamanhos aproximadamente iguais (*k-fold cross validation*). Com isso, o treinamento é realizado k vezes, a cada vez deixando um dos grupos para teste. Ou seja, se K = 5, a rede será treinada cinco vezes, na primeira vez o primeiro grupo será usado para teste e os outros quatro serão usados para treinamento. Na segunda vez, o segundo grupo será para teste e os outros quatro serão para treinamento, e assim sucessivamente.

A vantagem de se usar a validação cruzada ao invés do método *holdout* é que nela o treinamento é feito com todos os dados, e por isso gera um resultado mais confiável, uma vez que no método *holdout* os dados são divididos e essa divisão pode não gerar um resultado representativo dos padrões.

3. METODOLOGIA

Essa seção apresenta como serão as etapas do trabalho, quais os métodos utilizados e como ocorrerá as seleções.

3.1. Seleção de Repositório

A seleção do repositório ocorrerá por meio de uma análise dos repositórios disponíveis na web, com código aberto. Será necessário também analisar a relevância dos dados para alcançar o objetivo do trabalho, levando em conta as próximas etapas.

3.2. Seleção de Dados

Considerando que cada issue registrada possui características próprias, será necessário verificar o funcionamento das mesmas, os dados apresentados de acordo com o status e os campos presentes em suas características, afim de obter informações relevantes para sua classificação.

3.3. Mineração dos Dados

Em sequência, a mineração de dados será responsável por trabalhar nos dados obtidos anteriormente através de diversas técnicas disponíveis. A escolha dessas técnicas será baseada em uma análise das mesmas, afim de escolher as mais adequadas para este conjunto de dados.

3.4. *Machine Learning*

Nessa etapa, será necessário a escolha de possíveis classificadores e testá-los, afim de obter um modelo de classificação de *bugs*. A classificação será realizada de acordo com apenas uma característica de qualidade, sendo *bugs* de segurança ou não segurança.

Os *bugs* considerados de segurança possuem acessos restritos, afim de evitar possíveis ataques, uma vez que um ataque pode causar danos no software. Baseando-se nisso, a característica de segurança foi escolhida com o intuito de impossibilitar acessos não autorizados às informações confidenciais do software.

3.5. Resultados

Esta etapa consiste na análise e considerações de todas as informações relevantes, sendo elas a forma de realização da classificação, as técnicas de mineração de dados e os métodos utilizados no *machine learning*. Essas análises serão de extrema importância, afim de identificar a eficiência do resultado, a coerência da classificação de dados e também para identificar qual classificador atendeu melhor o conjunto de dados e as possíveis melhorias em um projeto futuro.

4. DESENVOLVIMENTO

Nesta seção será detalhada minunciosamente todas as etapas citadas na seção anterior.

4.1. Seleção de repositório

Existem vários repositórios disponíveis na *web*, dentre eles analisou-se repositórios com *bugs* resolvidos e devidamente fechados, pois assim é garantido que esses *bugs* já estejam classificados corretamente.

Dentre vários repositórios analisados foram escolhidos três principais: Mozilla, Gentoo e Eclipse. Eles foram selecionados devido a quantidade de bugs já resolvidos e os critérios de análise dos três foram aplicados com a utilização de buscas feitas através de filtros na pesquisa avançada, sendo eles:

- No repositório do Gentoo, os *bugs* pesquisados tiveram o campo *Product* com o valor *Gentoo Security*, o campo *Status* com o valor *Resolved* e o campo *Resolution* com valor *Fixed.* Sendo assim, os *bugs* são fechados, resolvidos e classificados como *bugs* de segurança. Essa busca resultou em 10.000 *bugs*.
- No repositório do Mozilla, o campo *Product* com todos os valores disponíveis, *Component Security*, *Status Resolved* e *Resolution Fixed* resultaram em 4550 *bugs*.
- No Eclipse, *Component Security* e *Security Management*, *Status Resolved* e *Resolution Fixed*, resultaram em 145 *bugs*.

Como citado acima, os campos *resolved* e *fixed* foram um dos critérios da análise para garantir que os *bugs* já estejam classificados corretamente, pois ao decorrer do processo de resolução dos *bugs*, a classificação pode ser mudada.

Em relação aos outros campos, cada repositório utiliza um campo especifico para identificar a classificação. No caso do *Gentoo* foi utilizado o campo *product* com valor *Gentoo Security* no qual identifica que o bug pertence a classificação de segurança.

No Mozilla, o campo *product* precisa que todos os valores sejam marcados para englobar todos os *bugs* relatados e o campo que define sua classificação é o *Component* com valor *Security.* Já o Eclipse segue semelhante ao Mozilla, porém com um *Component* a mais que é o *Security Management*.

Com essas informações, foi escolhido o Bugzilla do Gentoo para este trabalho, uma vez que sua base de dados apresentou uma quantidade maior de bugs, nos proporcionando uma melhor análise para classificação.

4.2. Seleção dos dados

Para a seleção de dados, os *bugs* foram analisados e realizou-se a busca avançada no Bugzilla, onde foram feitas algumas considerações. Dentre elas, podem ser citados os critérios utilizados na seleção do repositório do Gentoo. Além deles, no campo *Component* é utilizado todos os valores disponíveis na busca, exceto o valor *Kernel*, pois esse componente é irrelevante e pode impactar na classificação. Diante disso, baseado nas características de um bug já classificado como de segurança, foram identificadas as características dos *bugs* de não segurança.

Todas essas características citadas se tornaram a variável *parameters* e estão presentes no seguinte trecho do código (Figura 10).

Figura 10 - Seleção de Dados

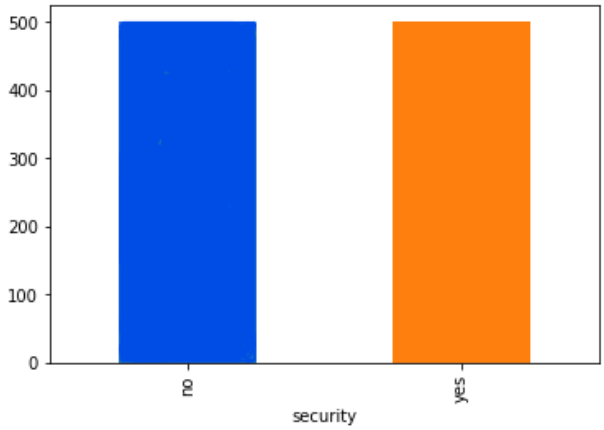
```
In [3]: base_url = 'https://bugs.gentoo.org/rest/bug'
        include_fields = ['id','summary']
        f1 ="product"
        f2 ="component"
        o1="equals"
        o2 ="notequals"
        query_format="advanced"
        v1="Gentoo Security"
        v2="kernel"
        limit=0
        order= "changeddate"

        parameters = {'include_fields': include_fields,
                      'f1' : f1,
                      'f2' : f2,
                      'o1' : o1,
                      'o2' : o2,
                      'query_format' : query_format,
                      'v1' : v1,
                      'v2' : v2,
                      'limit' : limit,
                      'order' : order
                     }

        r = requests.get(base_url, parameters)
        bugs = r.json()["bugs"]

        comments_url = "https://bugs.gentoo.org/rest/bug/#/comment?include_fields=text"
        separator = "\n"
```

Fonte: Do autor (2020)

Para a realização da classificação dos *bugs*, essas características foram utilizadas no momento de selecionar os dados relevantes, e foram passadas por parâmetros. Obteve-se uma base de dados inicial com 500 *bugs* solucionados, nos quais foram classificados sendo de segurança e 500 *bugs* de não segurança o qual não possuíam os critérios acima. Com as

funcionalidades disponíveis na biblioteca panda e na *matplotlib*, foi plotado um gráfico com os *bugs* de segurança e não segurança, como pode ser visualizado na Figura 11.

Figura 11 - *Bugs* de Segurança e Não Segurança



Fonte: Do autor (2020)

Essas bases de dados foram obtidas no formato CSV identificado como planilha e se faz necessária para realizar a classificação na etapa de *Machine learning*. A seleção dos *bugs* foi realizada por meio de comando *python*, utilizando o protocolo *Restful*, no qual pega a url do repositório e os parâmetros definidos. Assim é possível selecionar os dados e carregar as bases, aplicando o comando descrito na Figura 12.

Figura 12 - Comando para seleção das bases

```
r = requests.get(base_url, parameters)
bugs = r.json()["bugs"]
```

Fonte: Do autor (2020)

4.3. Mineração de dados

Após a obtenção da base dos dados, foi realizado um trabalho de pré-processamento a fim de preparar os dados em um formato adequado para a classificação. Com o uso da biblioteca *spacy* foram aplicadas as operações de tokenização e *lemming*, além disso teve outras operações citadas abaixo:

As tags, definida como a forma de identificar cada *bug*, foram retiradas através do pacote disponível no *python* chamado *BeautifulSoup.* Além disso, com a utilização da biblioteca *re*, as urls presentes nos *bugs* foram substituídas pela palavra url, para padronizar todas as urls com apenas um termo. Isso foi realizado com o uso do comando *re.sub* que fez as substituições nas

urls e retira todas as pontuações, números e múltiplos espaços presente nos dados, visto que esses campos são irrelevantes para a classificação.

Por fim tokenizou os termos, ou seja, a base de dados se tornou um vetor em que cada palavra é um elemento desse vetor. Aplicou-se então *lemming* nas palavras que não são nomes próprios para identificar em qual classe gramatical elas se identificam e *lowercase* em todas as palavras para convertê-las em letras minúsculas. A Figura 13 mostra os comandos utilizados para realizar esses passos.

Figura 13 - Pré-processamento das bases

Fonte: Do autor (2020)

Após esse pré-processamento, foi necessário carregar as bases do arquivo CSV gerado na etapa de seleção dos dados e juntá-las. Essa atividade foi realizada com o uso da biblioteca pandas, o qual possui as funcionalidades *pd.read_csv* para ler os dados da planilha CSV e *pd.concat* para juntar as bases lidas. Cada arquivo CSV possuía 500 *bugs*, porém, com a junção das bases, obteve-se um arquivo de 1000 *bugs*. Realizando a remoção de valores nulos com a funcionalidade *df.dropna*, obteve-se um arquivo de tamanho 987, conforme o código descrito na Figura 14.

Figura 14 - Carregamento e Junção das bases de dados

Fonte: Do autor (2020)

Após todo esse trabalho realizado nos dados, dar-se-á início a etapa de *machine learning*.

4.4. *Machine Learning*

Os classificadores utilizados no presente trabalho foram o *RandomForest* e *DecisionTree*. No classificador *RandomForest*, utilizou o parâmetro *RandomState* no momento de classificar, sendo necessário também o *n_jobs* para identificar o número de testes a serem realizados para fazer a média dos resultados, as variáveis min_terms e max_terms identificam a quantidade mínima de termos e máxima para gerarem as *wordclouds*.

A função *TD-IDF* disponível na biblioteca *sklearn* é responsável assinalar a importância dos termos na base de dados com a frequência em que ele aparece. Isso é realizado com o uso do parâmetro n-grama no qual define o *f1* como o peso dos termos, utilizando unigramas (1,1) e unigramas com bigramas (1,2) como parâmetro, conforme mostra a Figura 15.

Figura 15 - Definição de parâmetros

```
In [5]: x_cols = ["summary", "description"]
        y_col = "security"

        min_terms = 5
        max_terms = 100
        step = 1

        cv_num = 10

        n_grams = [(1,1), (1,2)]
        criterion = ['entropy', 'gini']

        terms_qty = list(range(min_terms, max_terms+1, step))

        vectorizer = TfidfVectorizer(use_idf=True, preprocessor=text_pre_processing,
                                     token_pattern=r'[^\s]+', sublinear_tf=True, norm='l2')

        classifier = RandomForestClassifier(random_state=42, n_jobs=4)

        param_grid = [
            {'Vectorizer__ngram_range': n_grams, 'Classifier__criterion': criterion},
        ]
```

Fonte: Do autor (2020)

Esse parâmetro controla a aleatoriedade do estimador utilizado até encontrar a melhor divisão. Ressalta-se que a melhor divisão pode variar entre execuções diferentes, quando o número de recursos a serem considerados em cada divisão é menor que o número total de recursos. O algoritmo seleciona a melhor execução em cada divisão e depois os compara para encontrar a divisão mais apropriada. No entanto, quando esses valores são iguais, é necessário fixar esse parâmetro em um número inteiro para obter um comportamento determinístico já que os recursos são permutados aleatoriamente.

Para realizar a classificação, tornou-se necessário utilizar a pipeline em todo o texto, a fim de que, combinada com os classificadores utilizados, construa um estimador composto para ser utilizado na classificação.

Os parâmetros dos classificadores foram definidos com a variável *criterion.* Essa variável mede a qualidade da divisão, como também suporta a impureza de Gini e o ganho de informação. O classificador *DecisionTree* apresenta o parâmetro *splitter,* no qual suporta duas estratégias, a *best* e a *random.* A *best* para escolher a melhor divisão e a *random* para escolher a melhor divisão aleatória. Neste trabalho foi utilizada a *best*. Para selecionar os melhores parâmetros foi usada a função *score,* definida como função de pontuação usada nos dados retidos.

Para avaliação de desempenho foi utilizada a validação cruzada, com a base de teste e treino, com um conjunto total de 100 termos. Não foi utilizado todos os termos presente na base de dados para que não sobrecarregue as execuções, pois quanto mais termos mais poder computacional seria necessário para treinar e avaliar o modelo. Os recursos disponíveis para o presente trabalho são um pouco limitados, pois foi realizado em dispositivo pessoal usado para estudos, com isso fez-se necessário utilizar um conjunto menor de termos. Porém, aumentar essa quantidade de termos é uma possibilidade válida para se explorar caso tenha melhores recursos.

Foi utilizada a métrica F1 para obter os melhores resultados de cada classificador. Essa métrica pega o melhor resultado dos testes realizados. A pontuação é considerada uma média ponderada da precisão do classificador, no qual para atingir seu melhor valor a pontuação deve chegar em 1 e para atingir seu pior valor a pontuação deve chegar em 0.

A função *average_precision_score* foi utilizada no presente trabalho. Essa função calcula a precisão média por meio de uma curva de precisão e recuperação com a média ponderada alcançada em cada limite, de acordo com a fórmula descrita na Figura 16.

Figura 16 - Fórmula da função *average_precision_score*

$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$$

Fonte: *Scikit-Learn*

No qual R é definido como Revocação e P como a Precisão, apresentadas na Seção 2.3 (veja a Figura 7 Figura 8 respectivamente para mais detalhes). Essas quantidades também estão relacionadas à pontuação como a média harmônica de precisão e revocação, conforme a Figura 16.

De acordo com o código mostrado na Figura 17, o *f1* médio foi calculado e os resultados foram gerados conforme será apresentado na próxima subseção.

Figura 17 - Colhendo resultados

```
    result = {
        "terms_qty": n,
        "best_micro_f1": grid_search.best_score_,
        "best_params": grid_search.best_estimator_.get_params(),
        "best_terms": sorted(vocabulary.items(), key=lambda x: x[1], reverse=True)[:n],
    }


    avgs = grid_search.cv_results_['mean_test_score']

    tests = list()
    for i, n in enumerate(avgs):
        test = "Test-" + str(i)
        tests.append(test)
        result[test] = n

    results.append(result)

results_df = pd.DataFrame(results)
results_df = results_df.set_index("terms_qty")
try:
    results_df.to_csv("results" + str(terms_qty) + ".csv")
except:
    results_df.to_csv("results" + str(len(terms_qty)) + ".csv")
```

Fonte: Do autor (2020)

4.5. Resultados

Ao fim da classificação foi gerado várias *wordclouds* com os modelos analisados conforme citado na subseção anterior. Dentre eles, pode-se citar as *wordclouds* com o número mínimo, médio e máximo de elementos de cada classificador. Essas *wordclouds* são geradas com os termos já classificados e cada um deles possuem pesos diferentes, sendo que quanto maior o tamanho do termo, maior será o peso dele e quanto mais próximo de um, melhor será sua precisão, visto que ao chegar no peso 1 significa que se obteve 100% de precisão.

A Figura 18, 19 e 20 representa *Wordcloud* com 5, 50 e 100 elementos do melhor treinamento do classificador *Random Forest* respectivamente.

- Termo de peso máximo: *reproducible* (Pronome) - 0.01109116115090657
- Termo de peso mínimo: *reproduce* (Verbo) - 0.007157568507414262

Figura 18 - *Wordcloud* com 5 elementos do classificador *Random Forest*

Fonte: Do autor (2020)

- Termo de peso máximo: *vulnerability* (Substantivo) - 0.042844639584915 01
- Termo de peso mínimo: *validation* (Substanivo) - 0.003463279252346946

Figura 19 - *Wordcloud* com 50 elementos do classificador *Random Forest*

Fonte: Do autor (2020)

- Termo de peso máximo: *reproducible* (Pronome) - 0.0653301623127003
- Termo de peso mínimo: *author* (Substantivo) - 0.0019855151031882554

Figura 20 - *Wordcloud* com 100 elementos do classificador *Random Forest*

Fonte: Do autor (2020)

A Figura 21, 22 e 23 representa *Wordcloud* com 5, 50 e 100 elementos do melhor treinamento do classificador *Decision Tree* respectivamente.

- Termo de peso máximo: *glsa* (Pronome) - 0.021514607511732356
- Termo de peso mínimo: *address* (Verbo) - 0.014851053860269716

Figura 21 - *Wordcloud* com 5 elementos do classificador *Decision Tree*

Fonte: Do autor (2020)

- Termo de peso máximo: *attack* (Substantivo) - 0.043041690583724784
- Termo de peso mínimo: *by-pass* (Substantivo) -0.005547585781758508

Figura 22 - *Wordcloud* com 50 elementos do classificador *Decision Tree*

Fonte: Do autor (2020)

- Termo de peso máximo: *arbitrary* (Adjetivo) - 0.05653142553588641
- Termo de peso mínimo: r (PUNCT) - 0.002791422857409881

Figura 23 - *Wordcloud* com 100 elementos do classificador *Decision Tree*

Fonte: Do autor (2020)

As *wordclouds* são representações do peso de termos mais significativos e menos significativos que definem a classificação dos *bugs.* Através do peso de cada termo o classificador consegue calcular o peso geral com a quantidade de termos de cada teste, por isso a importância dela na classificação.

Mediante isso, pode-se notar que os elementos menos relevantes são os de menor quantidade. As *wordclouds* com 5 elementos apresentaram termos comuns que não são da área de segurança que se repetiram bastante nas *issues*, com isso não se pode concluir que um *bug* é realmente de segurança.

No entanto, as *wordclouds* com maiores quantidades de elementos apresentam termos mais relevantes, visto que tem ligação com a característica de segurança. Por exemplo, as palavras *attack* ou *vulnerable,* com isso, em outros repositórios é bem possível que os termos sejam semelhantes, sendo assim, os termos poderiam também aparecer em outros treinamentos de outros projetos além do Gentoo.

Porém, não usar a maior quantidade de dados não resulta necessariamente em um melhor resultado, essa variação ocorre devido ao *overfit*. Quando utilizamos os dados de treino, o modelo tem um desempenho muito bom, porém quando utilizamos os dados de teste o desempenho diminui. Isso ocorre pois o modelo aprendeu tão bem as relações existentes no treino, que acabou apenas decorando o que deveria ser feito, e ao receber as informações das variáveis preditoras nos dados de teste, o modelo tenta aplicar as mesmas regras decoradas, porém com dados diferentes esta regra não tem validade, e o desempenho é afetado.

Realizou-se um treinamento de 4 testes em cada conjunto de termos. Os melhores resultados desses testes são definidos na variável *best_micro_f1,* no qual apresenta a precisão dos classificadores utilizados. Esse treinamento foi realizado com a ferramenta *GridSearchCV,* usada para realizar diversas combinações de parâmetros, avaliá-las e armazená-las em um único objeto.

Com isso, nota-se a diferença dos pesos dos termos entre as *wordclouds* do melhor treinamento entre os 4 testes. Por exemplo, o termo *glsa* foi o termo com maior peso na *wordcloud* com 5 elementos, já na *wordcloud* com 50 foi o termo *attack*. Essa diferença se dá pela aleatoriedade do estimador com diversas combinações de parâmetros, conforme explicado na seção 4.4.

Porém, mesmo com a aleatoriedade e combinações de parâmetros, existem palavras que aparecem com pesos razoavelmente altos e sem relação relevante para classificação de um *bug* de segurança. Contudo, em um estudo futuro pode-se avaliar as combinações de parâmetros ou realizar uma quantidade maior de testes, a fim de melhorar os resultados de classificação dos *bugs* de segurança ou de qualquer outra característica.

Ao decorrer da Tabela 1 e Tabela 2, apresenta-se algumas variantes de classificação realizadas. Na primeira coluna nomeada de “terms-qty” apresenta-se a quantidade de termos que o classificador utilizou. Da segunda coluna até a quarta coluna, nomeadas de Test-0 até test-3, apresenta-se o resultado referente a classificação, sendo que cada um desse teste foram resultados de uma classificação.

Com isso, deve-se notar que quanto mais próximo de 1 o valor, melhor será o resultado, visto que se chegar a 1 o resultado alcança 100% de precisão. Na última coluna, nomeada de best_micro_f1, repete-se o maior valor da linha, indicando qual foi o melhor resultado obtido para a quantidade de termos. Assim, a leitura da primeira linha ficaria: Dentre os 4 testes realizados com 99 termos, o Test-2, com peso 0.844985, foi o que obteve o melhor resultado.

Tabela 1 - Amostragem com 60 termos dos resultados finais – *Decision Tree*

terms_qty	Test-0	Test-1	Test-2	Test-3	best_micro_f1
99	0.817629	0.791287	0.844985	0.785208	0.844985
98	0.845998	0.787234	0.841945	0.816616	0.845998
97	0.851064	0.792300	0.835866	0.800405	0.851064
96	0.828774	0.805471	0.829787	0.807497	0.829787
95	0.851064	0.798379	0.828774	0.783181	0.851064
94	0.833840	0.787234	0.837893	0.772036	0.837893
93	0.827761	0.803445	0.839919	0.787234	0.839919
92	0.841945	0.803445	0.815603	0.761905	0.841945
91	0.833840	0.812563	0.842958	0.792300	0.842958
90	0.830800	0.794326	0.803445	0.781155	0.830800
88	0.848024	0.800405	0.816616	0.798379	0.848024
87	0.833840	0.775076	0.789260	0.790274	0.833840
86	0.832827	0.788247	0.812563	0.814590	0.832827
85	0.838906	0.811550	0.815603	0.775076	0.838906
83	0.845998	0.772036	0.827761	0.791287	0.845998
82	0.820669	0.801418	0.840932	0.799392	0.840932
79	0.835866	0.784195	0.835866	0.762918	0.835866
78	0.838906	0.785208	0.834853	0.789260	0.838906
77	0.835866	0.785208	0.810537	0.790274	0.835866
76	0.824721	0.790274	0.830800	0.781155	0.830800
73	0.833840	0.778116	0.829787	0.790274	0.833840
70	0.841945	0.791287	0.816616	0.751773	0.841945
65	0.833840	0.778116	0.838906	0.784195	0.838906
64	0.830800	0.788247	0.799392	0.780142	0.830800
63	0.832827	0.779129	0.807497	0.774063	0.832827
62	0.811550	0.791287	0.834853	0.786221	0.834853

54	0.839919	0.772036	0.825735	0.787234	0.839919
52	0.831814	0.790274	0.823708	0.787234	0.831814
45	0.794326	0.759878	0.787234	0.753799	0.794326
44	0.830800	0.780142	0.775076	0.799392	0.830800
43	0.796353	0.781155	0.787234	0.794326	0.796353
41	0.799392	0.774063	0.800405	0.772036	0.800405
40	0.812563	0.766971	0.834853	0.776089	0.834853
38	0.806484	0.751773	0.792300	0.750760	0.806484
37	0.786221	0.758865	0.804458	0.727457	0.804458
35	0.803445	0.759878	0.795339	0.751773	0.803445
33	0.798379	0.767984	0.775076	0.761905	0.798379
32	0.777102	0.762918	0.802432	0.758865	0.802432
29	0.801418	0.761905	0.805471	0.747720	0.805471
28	0.794326	0.751773	0.792300	0.765957	0.794326
26	0.782168	0.773050	0.776089	0.743668	0.782168
25	0.780142	0.770010	0.775076	0.743668	0.780142
23	0.790274	0.768997	0.756839	0.763931	0.790274
21	0.779129	0.757852	0.772036	0.774063	0.779129
20	0.786221	0.766971	0.783181	0.764944	0.786221
19	0.774063	0.768997	0.763931	0.754813	0.774063
18	0.805471	0.746707	0.751773	0.728470	0.805471
17	0.798379	0.740628	0.786221	0.760892	0.798379
16	0.751773	0.758865	0.766971	0.753799	0.766971
15	0.784195	0.751773	0.764944	0.748734	0.784195
14	0.788247	0.781155	0.753799	0.754813	0.788247
13	0.754813	0.738602	0.773050	0.739615	0.773050
12	0.763931	0.739615	0.767984	0.716312	0.767984
11	0.751773	0.756839	0.744681	0.741641	0.756839
10	0.744681	0.730496	0.756839	0.742655	0.756839
9	0.753799	0.739615	0.753799	0.716312	0.753799
8	0.751773	0.716312	0.771023	0.755826	0.771023
7	0.782168	0.757852	0.752786	0.730496	0.782168
6	0.731510	0.724417	0.754813	0.734549	0.754813
5	0.764944	0.758865	0.737589	0.724417	0.764944

Fonte: Do autor (2020)

Tabela 2 - Amostragem com 60 termos dos resultados finais – *Random Forest*

terms_qty	Test-0	Test-1	Test-2	Test-3	best_micro_f1
99	0.916920	0.884498	0.919959	0.882472	0.919959
98	0.919959	0.875380	0.911854	0.879433	0.919959
97	0.929078	0.881459	0.910841	0.882472	0.929078
96	0.920973	0.878419	0.921986	0.878419	0.921986
94	0.917933	0.863222	0.915907	0.882472	0.917933
93	0.924012	0.879433	0.915907	0.869301	0.924012

92	0.913880	0.874367	0.916920	0.891591	0.916920
91	0.927052	0.869301	0.919959	0.872340	0.927052
89	0.924012	0.877406	0.913880	0.862209	0.924012
87	0.911854	0.865248	0.918946	0.877406	0.918946
86	0.916920	0.869301	0.917933	0.869301	0.917933
84	0.918946	0.852077	0.915907	0.862209	0.918946
83	0.912867	0.867275	0.917933	0.865248	0.917933
81	0.913880	0.885512	0.918946	0.879433	0.918946
80	0.910841	0.867275	0.917933	0.852077	0.917933
79	0.905775	0.863222	0.921986	0.867275	0.921986
78	0.917933	0.894630	0.904762	0.870314	0.917933
77	0.925025	0.889564	0.928065	0.868288	0.928065
76	0.921986	0.871327	0.914894	0.868288	0.921986
75	0.912867	0.870314	0.928065	0.869301	0.928065
74	0.918946	0.883485	0.912867	0.872340	0.918946
71	0.917933	0.865248	0.924012	0.848024	0.924012
69	0.928065	0.878419	0.915907	0.881459	0.928065
66	0.918946	0.865248	0.916920	0.855117	0.918946
64	0.931104	0.855117	0.915907	0.835866	0.931104
63	0.908815	0.858156	0.918946	0.845998	0.918946
62	0.921986	0.868288	0.905775	0.869301	0.921986
58	0.902736	0.862209	0.916920	0.860182	0.916920
56	0.913880	0.860182	0.931104	0.862209	0.931104
55	0.930091	0.859169	0.910841	0.867275	0.930091
54	0.900709	0.872340	0.894630	0.861196	0.900709
52	0.899696	0.863222	0.902736	0.874367	0.902736
41	0.895643	0.864235	0.895643	0.857143	0.895643
40	0.895643	0.873354	0.895643	0.860182	0.895643
38	0.895643	0.858156	0.899696	0.860182	0.899696
35	0.898683	0.842958	0.898683	0.855117	0.898683
34	0.895643	0.852077	0.898683	0.834853	0.898683
32	0.887538	0.851064	0.901722	0.830800	0.901722
31	0.899696	0.844985	0.880446	0.852077	0.899696
29	0.885512	0.847011	0.883485	0.844985	0.885512
25	0.883485	0.849037	0.887538	0.850051	0.887538
24	0.890578	0.845998	0.885512	0.844985	0.890578
23	0.885512	0.839919	0.888551	0.851064	0.888551
22	0.900709	0.853090	0.890578	0.837893	0.900709
21	0.891591	0.847011	0.880446	0.829787	0.891591
20	0.881459	0.840932	0.894630	0.844985	0.894630
19	0.881459	0.851064	0.883485	0.843972	0.883485
18	0.887538	0.854103	0.893617	0.845998	0.893617
17	0.879433	0.823708	0.878419	0.806484	0.879433
16	0.870314	0.816616	0.887538	0.796353	0.887538
15	0.874367	0.839919	0.866261	0.847011	0.874367
13	0.887538	0.829787	0.881459	0.835866	0.887538

12	0.881459	0.829787	0.865248	0.832827	0.881459
11	0.890578	0.819656	0.878419	0.822695	0.890578
10	0.871327	0.834853	0.858156	0.819656	0.871327
9	0.875380	0.822695	0.866261	0.849037	0.875380
8	0.875380	0.823708	0.878419	0.831814	0.878419
7	0.870314	0.808511	0.878419	0.852077	0.878419
6	0.879433	0.830800	0.864235	0.814590	0.879433
5	0.863222	0.839919	0.858156	0.813576	0.863222

Fonte: Do autor (2020)

Conforme os resultados obtidos, os modelos analisados apresentaram melhores resultados quando utilizado o classificador *Random Forest*. Pode-se perceber que os termos mais significativos tiveram uma classificação maior considerando substantivos.

Pode-se notar também que quanto maior a quantidade de termos, mais significativo o peso dos termos ficava. As taxas de desempenho apresentaram uma diferença relativa sobre as quantidades de termos. Nota-se essa observação nos resultados das *wordclouds* com número mínimo e máximo de termo. No classificador *Random Forest*, o termo com peso máximo foi de 0.01109116115090657 para a w*ordcloud* com 5 elementos, enquanto o termo de peso máximo foi de 0.0653301623127003 para o mesmo classificador, porém com 100 elementos. Já no *Decision Tree*, a *wodcloud* com 5 elementos apresentou um resultado de 0.021514607511732356 para o termo com peso máximo e a *wodcloud* com 100 elementos apresentou um resultado de 0.05653142553588641 para o termo com peso máximo.

Além dessas diferenças no peso de cada termo com quantidades distintas, é possível notar as diferenças nos classificadores também. Foram apresentados quatro testes e qual foi o melhor resultados dentre eles. As variações de resultados ocorreram por meio da quantidade de termos usados. Os melhores resultados nos dois classificadores se deram com uma quantidade maior de termos, isso pode ser justificado pelo fato de ser um número maior de informações sobre os *bugs*.

Contudo, o melhor resultado dos testes dos dois classificadores foram:

- *Random Forest* com 64 termos:
    - Teste 0: peso de 0.931104;
    - Teste 1: peso de 0.855117;
    - Teste 2: peso de 0.915907;
    - Teste 3: peso de 0.835866;
    - Micro f1: peso de 0.931104;

- *Decision Tree* com 97 termos:
    - Teste 0: peso de 0.851064;
    - Teste 1: peso de 0.792300;
    - Teste 2: peso de 0.835866;
    - Teste 3: peso de 0.800405;
    - Micro f1: peso de 0.851064;

Portanto, de acordo com os resultados apresentados, foi comprovado que o classificador *Random Forest* apresentou o melhor resultado na classificação automática de *bugs* referente ao *Bugzilla* do *Gentoo*, pois sua média apresentou 93% de precisão na classificação.

Pode-se notar também que os melhores resultados dos dois classificadores foram encontrados na execução do Teste 0.

5. CONCLUSÃO

O presente trabalho abordou sobre a classificação automática de *bugs* no repositório do Gentoo, afim de aumentar a qualidade do software seguindo os critérios estabelecidos pela norma ISO IEC 2500. Através da classificação automática muitos *bugs* poderão ser resolvidos com maior agilidade no tempo de correção e menor custo da triagem dos erros, além de ser distribuído de acordo com suas características para os especialistas adequados de cada área.

O tema abordado foi de suma importância para minha carreira profissional, visto que já atuo na área de qualidade de software profissionalmente e por meio desse trabalho foi possível ter maior conhecimento sobre a área.

A classificação foi realizada com a utilização de dois algoritmos de *Machine Learning*, sendo eles: *Random Forest* e *Decision Tree*, coletando os *bugs* do repositório e classificando-os em *bugs* de segurança e não segurança. Os resultados obtidos permitem concluir que o classificador *Random Forest* apresentou um melhor resultado tanto na fase de treinamento quanto na média do resultado final para este conjunto de dados analisados.

Contudo, o objetivo do trabalho de implementar um protótipo para a classificação foi alcançado, apresentando os resultados citados acima. É possível dar prosseguimento à pesquisa, incrementando ao protótipo outros algoritmos de *Machine Learning* a fim de avaliar qual apresenta melhor resultado de acordo com o conjunto de dados. Além disso, pode-se escolher um conjunto de dados maior para classificação e aumentar a classificação de acordo com as oito características de qualidade.

6. REFERENCIAS

AMO, S. **Técnicas de mineração de dados**. Universidade Federal de Uberlândia, Faculdade de Computação, disponível em http://www.deamo.prof.ufu.br/, 2003.

Breiman, L. (2001a). Statistical modeling: the two cultures.Statistical Sci-ence, 16, 199-215.

Broder, Andrei Z.; Glassman, Steven C.; Manasse, Mark S.; Zweig, Geoffrey (1997). "Syntactic clustering of the web". Computer Networks and ISDN Systems. 29: 1157–1166.

CARVALHO, André et al. Inteligência Artificial: Uma abordagem de Aprendizado de Máquina. Rio de Janeiro: Ltc, 2011.

Chakure, A. (2019). Random Forest and its Implementation. Retrieved 5 December 2019. Disponível em: <https://towardsdatascience.com/random-forestand-its-implementation-71824ced454f>. Acesso em 19 de out. 2019

CORRÊA, Ulisses. **Mineração de Dados de Help Desk Usando Rattle – O Caso Petrobras**. 2007. 92 p. Dissertação (Mestrado) – Programa de Pós-Graduação e Pesquisa em Administração e Economia, Faculdade de Economia e Finanças IBMEC. Rio de Janeiro. 2007.

DE LUCCA, JL; NUNES, Maria das Graças Volpe. Lematização versus Stemming. USP, UFSCar, UNESP, São Carlos, São Paulo, 2002. Citado 1 vez na página 12.

DELEN, D. et al. Predicting breast cancer survivability: a comparison of three data mining methods. Artificial Intelligence in Medicine, In Press, Corrected Proof, Available online 11 September 2004.

DERCZYNSKI, Leon et al. Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data. In: PROCEEDINGS of the International Conference Recent Advances in Natural Language Processing RANLP 2013. Hissar, Bulgaria: INCOMA Ltd. Shoumen, BULGARIA, set. 2013. p. 198–206. Disponível em: . Citado 1 vez na página 12.

DINIZ, F., Neto, F. M., Júnior, F. d. C. and Fontes, L. M. (2013). **Redface:** um sistema de reconhecimento facial baseado em técnicas de análise de components principais e autofaces, Revista Brasileira de ComputaçãoAplicada 5(1): 42–54.

DUDA, R. O., HART, P. E. e STORK, D. G. Pattern Classification. Wiley- Interscience, second edition, 2000.

FAYYAD, U; PIATETSKY-SHAPIRO, G; SMYTH, P. **From data mining to knowledge discovery in databases**. Ai Magazine, 17:37–54, 1996.

GOMES, Helder Joaquim Carvalheira. **Text Mining**: análise de sentimentos na classificação de notícias. Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on. Lisboa. 2013.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. Data Mining Concepts and Techniques. 3º Edition. Elsevier, 2012.

HASTIE, T., Tibshirani, R. and Friedman, J. **The Elements of Statistical Learning**: Data Mining, Inference, and Prediction,Second Edition , Springer Series in Sta- tistics, Springer. (2009).

HEARST, M. A. Untangling text data mining. **Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics** (pp. 3-10),1999. Stroudsburg, PA, USA: Association for Computational Linguistics.

ISO 25000 *Standards* (2020) – (https://iso25000.com/index.php/en/iso-25000-standards)

ISO/IEC 2510 *Standard (2010)* (https://www.iso.org/standard/35733.html).

KIDO G.S. e JUNIOR S.B., Comparação entre TF-IDF e LSI para pesagem de termos em micro-blog, página 2.

K. A. Vidhya and G. Aghila, "**A Survey of Naïve Bayes *Machine learning* approach in Text Document Classification**", International Journal of Computer Science and Information Security, vol. 7, no. 2, (2010), pp. 206-211.

L. Wang, X. Zhao, "**Improved knn Classification Algortihm Research in Text Categorization**", In the Proceedings of the 2nd International Conference on Communications and Networks (CECNet), (2012), pp. 1848-1852.

LOPES, M. C. S. **Mineração de Dados Textuais Utilizando Técnicas de Clustering para o Idioma Português**. PhD thesis, Universidade Federal do Rio de Janeiro, 2004.

LYU, M. R. Handbook of Software Reliability Engineering. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

MAUDA, Everton Carlos. **Modelo de qualidade para características internas de segurança de componentes de software**. Dissertação, Universidade Católica do Paraná, 2012.

MITCHELL, T. Machine learning. New York: McGraw-Hill. 1997

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre Aprendizado de Máquina. In: MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Sistemas Inteligentes Fundamentos e Aplicações. Barueri, Sp: Manole Ltda, 2003. p. 39-56.

MOURA, M. F. **Proposta de utilização de mineração de textos para seleção, classificação e qualificação de documentos**. Embrapa Informática Agropecuária, 2004, ISSN 1677-9274, 2004.

Ramos, J. (2003). "Using TF-IDF to Determine Word Relevance in Document Queries". Technical report, Department of Computer Science, Rutgers University, 23515 BPO Way, Piscataway, NJ, 08855e.

RIBEIRO, L. N. Rede Neural com Retropropagação: uma Aplicação na Classificação de Imagens de Satélite. Dissertação de mestrado. Dezembro, 2003.

TAN, Ah-Hwee. **Text mining**: The state of the art and the challenges. In: PROCEEDINGS OF THE PAKDD 1999 WORKSHOP ON KNOWLEDGE DISCOVERY FROM ADVANCED DATABASES, Beijing. 1999.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. **Introdução ao Data Mining Mineração de Dados**. Rio de Janeiro: Editora Ciência Moderna. 2009.

THEODORIDIS, S. e KOUTROUMBAS, K. Pattern Recognition. Elsevier, second edition, 2003.

Verma T. e Renu, G.D. (2014). Tokenization and Filtering Process in RapidMiner. International Journal of Applied Information Systems (IJAIS). Volume 7 - No2, páginas 16-18.

A.C.E.S. Lima e L.N de Castro. (2012). "Automatic sentiment analysis of twitter messages". In Computational Aspects of Social Networks. Fourth International Conference on. Páginas 52-57.

G. Jeong, S. Kim, and T. Zimmermann, "**Improving bug triage with bug tossing graphs**," in Proc 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009, pp. 111–120.

T. M. Mitchell, "***Machine learning***", Carnegie Mellon University, McGraw-Hill Book Co, (1997).