RAQUEL HENGEN RIBEIRO

Identificacão de bugs em código-fonte usando aprendizagem de máquina

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Guilherme Dal Bianco
Este trabalho de conclusão de curso foi defendido e aprovado pela banca em: 01/09/2021

BANCA EXAMINADORA:

_____
Dr. Guilherme Dal Bianco - UFFS

_____
Dr. Denio Durte - UFFS

_____
Dr. Geomar Andre Schreiner - UFFS

Identificação de *bugs* em código-fonte usando aprendizagem de máquina

Raquek Hengen Ribeiro[1]

[1]Universidade Federal da Fronteira Sul - UFFS

raquel.hengen.ribeiro@gmail.com

Abstract. *During the life cycle of a software it is common to experience some type of system failure, originated from bugs in source code. Finding bugs is an extremely complex and onerous task. An alternative to reduce this work is to use a vulnerability prediction model (VPM). For the construction of an MPV it is necessary to extract characteristics from the source code to be applied in the prediction model. This work presents an experimental analysis of active learning techniques applied in a VPM. The experiment shows that the same active learning techniques used to reduce effort in literature reviews, can detect about 97% of vulnerable files, inspecting about 22% of files.*

Resumo. *Durante o ciclo de vida de um software é comum ocorrer algum tipo de falha de sistema, originadas de bugs em código-fonte. Encontrar esses bugs é uma tarefa extremamente complexa e onerosa. Uma alternativa para diminuir o esforço é usar um modelo de previsão de vulnerabilidade (MPV). Para a construção de um MPV é necessário extrair características sobre o código-fonte para serem aplicadas no modelo de predição. Este trabalho tem como objetivo uma análise experimental de técnicas de aprendizado ativo aplicadas como um MPV. O experimento mostra que as mesma técnicas de aprendizado ativo usados para redução de esforço nas revisões de literatura, podem detectar cerca de 97% dos arquivos vulneráveis, inspecionando cerca de 22% dos arquivos.*

1. Introdução

O ciclo de vida de um *software* consiste basicamente em projeto, desenvolvimento e manutenção. Conforme Pressman and Maxim (2016), durante a fase de projeto e desenvolvimento é construído toda a estrutura do código-fonte, implementando a revisão de código e aplicando diversos testes. Porém, na fase de manutenção nem sempre é possível testar e revisar código na sua completude. Conforme Pressman and Maxim (2016), cerca de 75% da vida útil de um *software* é gasto em manutenção.

Muitas falhas de sistema surgem no ciclo de vida de um *software*, algumas são, por exemplo, resultados inesperados do sistema, outras podem ser geradas devido a tentativa de invasão. Essas falhas são originadas por defeitos em código-fonte [Pfleeger 2004]. Conforme Shamal (2017), muitos dos defeitos em um código-fonte de um *software* podem levar a uma vulnerabilidade de sistema. Uma vulnerabilidade é uma fraqueza que permite que um atacante possa causar alguma incoerência ou roubo de dados. Para um sistema ser seguro, em teoria não deveria existir nenhum defeito em código. Porém, isso é uma tarefa complexa, devido a impossibilidade identificar todos os casos de usos e possibilidades. Na tentativa de tornar o software mais seguro, é feita a revisão de código. Nesta etapa,

uma equipe irá analisar arquivos de códigos-fonte com intuito de procurar por defeitos no software [Shamal et al. 2017].

Com o avanço da tecnologia, os códigos-fonte de *softwares* têm ficado cada vez mais complexos, por exemplo o conjunto do *Mozilla Firefox* tem mais de 28 mil arquivos de código-fonte. Analisar um conjunto desse porte manualmente representa uma tarefa inviável e onerosa. Uma alternativa, é utilizar Modelos de Previsão de Vulnerabilidade (MPV) cujo objetivo é detectar arquivos de código-fonte que contenham alguma vulnerabilidade.

O MPV é baseado em aprendizagem de máquina e o treinamento é feito a partir de erros conhecidos, métricas de *software* e código-fonte. Sabendo qual arquivo é vulnerável, o desenvolvedor pode focar na resolução do possível erro [Yu and Menzies 2018]. Os MPVs tem se mostrado cada vez mais precisos na identificação de vulnerabilidades, no entanto ainda é uma tarefa complexa e possui diversas limitações [Shamal et al. 2017]. Por exemplo, se o conjunto de códigos não possuir uma sinalização da ocorrência de um possível *bug*, será necessário o auxílio de um humano para validar uma grande quantidade de arquivos de código. Uma alternativa de diminuir o esforço do revisor humano é usando abordagens de aprendizado ativo [Settles 2009]. A aprendizagem ativa usa diversas técnicas para diminuir o esforço para solicitar o rótulo, ou seja, tenta ”prever” qual é o rótulo do documento antes de solicitá-lo [Settles 2009]. O método *knee* de Li and Kanoulas (2020), por exemplo, usa aprendizado ativo para determinar quais documentos devem ser rotulados pelo usuário. Além de diminuir o esforço de rotulação, o método estima o número de documentos relevantes da coleção, podendo interromper o método antes de analisar todos os documentos com segurança.

Neste trabalho é proposto uma análise experimental a fim de avaliar a extração de *features* e a configuração do método *Knee* no contexto do MPV. Dessa forma, o objetivo será explorar o uso do aprendizado ativo na identificação de arquivos de código-fonte que contêm algum tipo de vulnerabilidade.

Para a experimentação, a base de dados utilizada é oriunda do projeto *Mozilla Firefox*, contendo código aberto e com uma rotulação nos arquivos que contém algum tipo de *bug*. Foi observado que usando as *features* e configurações adequadas, o método *Knee* pode diminuir o esforço na revisão de código-fonte.

Este trabalho está estruturado na seguinte maneira: Na seção 2 é apresentada a fundamentação teórica envolvendo os conceitos básicos para o entendimento deste trabalho. A seção 3 trata de trabalhos semelhantes a este. A seção 4 apresenta os experimento desenvolvidos, e por fim a seção 5 trás as conclusões obtidas a partir deste trabalho.

2. Fundamentação Teórica

Nesta seção, são apresentados alguns métodos e conceitos para o entendimento do trabalho aqui proposto. Os conceitos são relacionados a engenharia de *software* e recuperação de informações.

2.1. Aprendizado de Máquina

O aprendizado de máquina tem como objetivo aprender com os dados sem ser explicitamente programado. A partir dos dados é feita uma tomada de de-

cisão [Duarte and Ståhl 2019]. Os modelos de aprendizagem de máquina podem ser divididos em dois grupos principais: o supervisionado e o não-supervisionado [Corcovia and Alves 2019]. No aprendizado supervisionado é dado um conjunto de entradas (atributos), e sua respectiva saída (rótulo). Este modelo tem como objetivo aprender uma regra generalizada na qual mapeia as entradas para as saídas de entradas não rotuladas. Já no aprendizado não-supervisionado, o arquivo de entrada não contém rótulos, então as *features* são combinadas em grupos (*clusters*) conforme sua similaridade [Corcovia and Alves 2019].

Uma das técnica de aprendizado de máquina supervisionado é a regressão logística, na qual busca estimar a probabilidade da variável dependente assumir um determinado valor em função dos valores conhecidos de outras variáveis [Dreiseitl and Ohno-Machado 2002]. A probabilidade de um exemplo pode ser definida pela seguinte Equação 1.

$$P(1|x, \alpha) = \frac{1}{1 + e^{-(\alpha.x)}} \tag{1}$$

Outra técnica de aprendizado de máquina supervisionado é a máquina de vetores de suporte (SVM). A SVM constrói uma linha de separação entre as classes dos conjuntos de dados [Dreiseitl and Ohno-Machado 2002]. O SVM pode ser ilustrado pela Figura 1. Na qual o define a classe "A-", e x define a classe "A+", e a linha pontilhada w define a linha de separação das classes de saída.

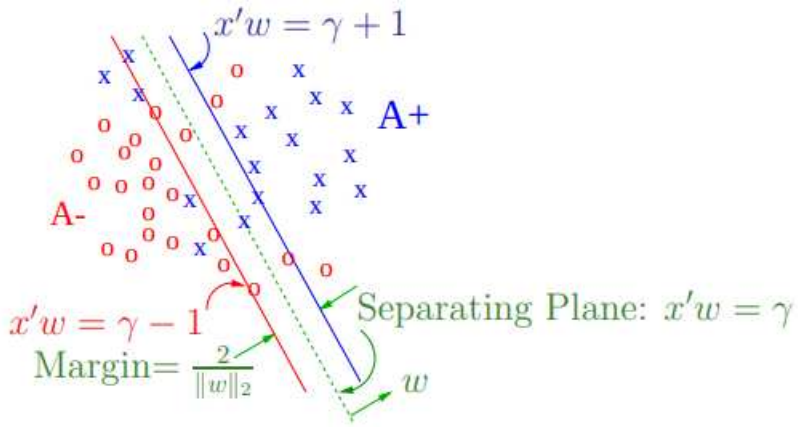

Figura 1. Ilustração da SVM
Fonte: [Mangasarian and Musicant 2000, p.3]

2.2. Extração de *features*

Os modelos de aprendizagem detectam padrões nos dados a partir de fórmulas matemáticas. Documentos textuais fazem parte de dados não estruturados, para conseguir aplicá-los num modelo de aprendizado de máquina é necessário estruturá-los [Aranha and Passos 2006].

Um dos métodos para estruturar o texto é o *Bag-of-words* (BOW). Na abordagem BOW cada documento de texto é representado por um vetor de ocorrências de palavras,

em outras abordagens usam frases ao invés de palavras. As palavras são separadas de formas distintas montando um vocabulário, após ter o conjunto de todas as palavras presentes nos documentos, é contado o número de ocorrência das palavras em cada documento [Li et al. 2010]. No entanto, o BOW não leva em consideração a importância dos termos, podendo prejudicar a análise das informações.

Para contornar as deficiências do BOW, foi proposto o TF-IDF (termo de frequência inversa). A ideia do TF-IDF é que se palavra ocorre com muita frequência em muitos documentos essa palavra é irrelevante (comum na maioria dos documentos). Ou seja, o objetivo é encontrar padrões a partir de informações relevantes e com diferença de comportamentos [Christian et al. 2016]. O termo de frequência tf_{td} é o percentual que o termo aparece no documento. Calculado pela seguinte Equação 2:

$$tf_{td} = \frac{tf_{t,d}}{|d|} \tag{2}$$

Na qual tf_{td} denota a frequência do termo t no documento d, $|d|$ representa o número total de termos presentes no documento. Após calcular a frequência dos termos, é necessário calcular a frequência inversa do documento idf_{td} para cada termo, ou seja, quantos documentos tem o mesmo termo. Calculada pela Equação 3.

$$idf_{td} = log\frac{|D|}{df_t} \tag{3}$$

Na qual D representa a coleção completa de documentos e df_t representa a frequência que o termo t aparece nos documentos. Finalmente temos a fórmula final em Equação 4:

$$tf - idf = tf_{td} \times idf_{td} \tag{4}$$

Na qual tf_{td} representa o termo de frequência, e idf_{td} representa a frequência inversa do termo, o resultado é o termo de frequência inversa do documento.

2.3. Recuperação de informação

O problema da recuperação de informação vem da necessidade de informação do usuário, ou seja, o usuário deseja encontrar documentos relevantes com sua pesquisa rapidamente sem precisar analisar previamente documentos irrelevantes. Para encontrar documentos relevantes com a pesquisa é necessário atingir um alto *recall*. Um rótulo humano pode melhorar o *recall* pois o computador não entende um texto desestruturado [Cleverdon 1974]. Uma abordagem usada na recuperação de informações, são técnicas de aprendizagem ativa.

Os sistemas de aprendizado ativo tem como objetivo diminuir o esforço para identificar os rótulos. Nos métodos ativos são solicitados os rótulos para um oráculo (por exemplo um inspetor humano). A aprendizagem ativa precisa focar em quais exemplos irá solicitar o rótulo para o oráculo. Uma técnica utilizada para isso é a amostragem de incerteza, que consiste em criar rótulos temporários. O rótulo é solicitado para as instâncias que estiverem mais próximas do plano de decisão (as que o algoritmo está incerto) [Settles 2009].

2.4. Modelo de previsão de vulnerabilidade

O modelo de previsão de vulnerabilidade (MPV) é baseado na aprendizagem de máquina e o treinamento é feito a partir de erros conhecidos (pode ser do próprio sistema ou de uma base de dados). Um MPV pode ser baseado em mineração de texto, métricas de *software* ou combinando ambos.

Métricas de código-fonte expressam características numéricas sobre o código-fonte. Algumas das métricas mais usadas são de tamanho e complexidade [Morais et al. 2012].

Em um MPV baseado em métricas de *software* as mesmas usadas no treinamento. Testes usando esse modelo conseguiram encontrar cerca de 75% das vulnerabilidades [Shamal et al. 2017].

Já o modelo baseado em mineração de texto usa técnicas de aprendizado de máquina e converte os *tokens* da linguagem em ocorrência de frequência. A ideia para este modelo é verificar os *tokens* que aparecem com menos frequência pois pode ser que tenha algo errado. Testes usando esse modelo conseguiram cerca de 80% de precisão [Shamal et al. 2017].

Para melhores resultados o MPV precisa ter um rótulo nos arquivos que contenham alguma vulnerabilidade. Algumas aplicações de código-aberto como o *Mozilla* contém uma sinalização em arquivos que contém algum tipo de *bug*. Caso a base de dados não contenha essa informação, é recomendado que um inspetor humano atribua os rótulos para os trechos de código [Yu et al. 2019].

2.5. KNEE

Algoritmos de aprendizado ativo demonstraram desempenho superior em comparação com outros métodos na identificação eficiente de documentos relevantes em medicina empírica e dados jurídicos. Um dos principais desafios para algoritmos de aprendizado ativo é decidir quando parar de exibir documentos aos revisores. O trabalho de Li and Kanoulas (2020) tem um objetivo duplo, primeiro identificar os documentos relevantes para revisão de literatura, e segundo estimar com precisão o número total de documentos relevantes para que interrompa o processo.

O método *knee* é o método mais promissor para a tarefa de *recall* total [Li and Kanoulas 2020]. O método *knee* (joelho) é um método guloso projetado por Satopaa et al. (2011) para atingir parada precoce. O ponto de parada do *knee* é baseado na noção dos pontos de curvatura máxima em um conjunto de dados. A Figura 2 mostra um joelho, o eixo Y representa alguma métrica de desempenho e o eixo X parâmetros ajustáveis. O ponto que está marcado representa o ponto que atinge curvatura máxima, ou seja, o ponto onde mais aprendeu [Satopaa et al. 2011].

O trabalho Li and Kanoulas (2020) apresenta uma combinação da parada precoce do *knee*, combinado com um ranqueamento dos documentos, assim satisfazendo seu objetivo. O ranqueamento é feito a partir de um algoritmo de aprendizagem de máquina.

O ciclo do *knee* é ilustrado na Figura 3 e detalhado a seguir:

1. **Extração de *features*:** A entrada do método é a coleção completa de documentos Dq e o tópico de busca. O tópico pode ser uma consulta de usuário com um

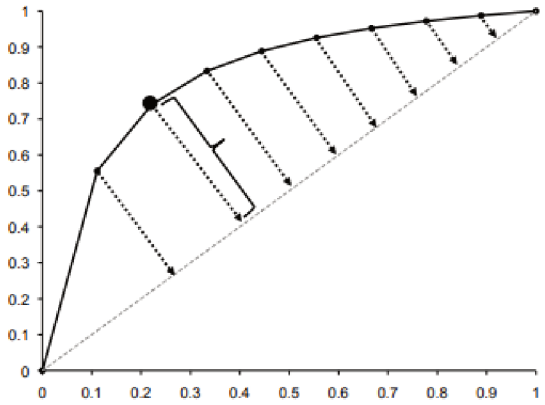

Figura 2. Detecção do Knee
Fonte: [Satopaa et al. 2011, p.3]

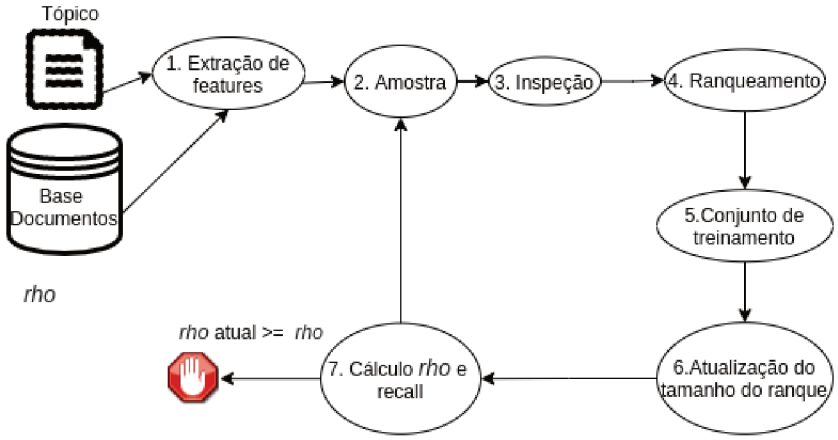

Figura 3. Detecção do Knee

documento relacionado a busca, ou apenas um documento no qual o objetivo é encontrar os relacionados. Também na entrada temos o *rho* de parada, que representa qual é o ponto de curvatura máxima. A extração de *features* é feita a partir do método TF-IDF, nesta extração são removidas as *stop words* que são as palavras que aparecem com muita frequência e podem causar ruído no método;

2. **Amostra:** Na primeira iteração do método é realizada uma amostra aleatória,e combinada com o tópico de entrada. Nas demais iterações, a amostra aleatória é feita a partir dos documentos que estão em Tu (documentos não rotulados);
3. **Inspeção:** Os documentos gerados na amostra aleatória passam pela inspeção de um revisor, no qual é atribuído seu respectivo rótulo (relevante 1 ou não relevante 0);
4. **Ranqueamento:** O ranqueamento é feito em duas etapas. A primeira etapa é o treinamento, que é feito a partir amostra aleatória juntamente com os documentos Rd (documentos do ranque). Esse treinamento é realizado com o algoritmo de aprendizagem de máquina Regressão Logística. Segunda etapa é o ranqueamento,

os todos os documentos da coleção passam por uma predição no mesmo algoritmo do treino, e são retornados os k melhores documentos relevantes Rd;
5. **Conjunto de Treinamento:** Adiciona os elementos treinados numa lista Lt, remova os elementos treinados de Tu (elementos que ainda não foram rotulados);
6. **Atualização do tamanho do ranque:** Aumente o tamanho do ranque, $k = k + k/10$;
7. **Cálculo *rho* e *recall*:** Realiza o cálculo do *rho* e do *recall*. Se o *rho* atual é maior ou igual ao *rho* desejado pare o método, se não volte a etapa 2.

No experimento de Li and Kanoulas (2020) para bases de dados textuais do EMED, TR, LEGAL o *knee* alcançou um *recall* que vária de 96,6% à 99,8 %, com um custo variando de 1,6% à 42% .

3. Trabalhos relacionados

A seguir neste capítulo será descrito os principais trabalhos relacionados a este projeto. Primeiro é apresentado um trabalho relacionado a recuperação de informações com objetivo de diminuir o esforço em revisão de literatura. Depois será apresentado dois trabalhos relacionados a modelo de previsão de vulnerabilidade (MPV). Destes o último é um MPV usando aprendizagem ativa.

Realizar uma revisão de literatura é uma tarefa onerosa. Buscadores como *Google Scholar* fazem uma pesquisa por palavras-chaves, e geralmente retornam milhares de resultados, muitos deles são irrelevantes, cabendo ao usuário revisar todos esses resultados manualmente. Para reduzir o esforço nas revisões bibliográficas na área de engenharia de *software* foi proposto um método de aprendizagem ativa [Yu et al. 2018]. Os conjuntos de dados usados nesse trabalho são: Wahono, Hall, Radjenovic e o Kitchenham.

O método proposto por Yu et al. (2018) o *FASTREAD* tem como foco diminuir os esforços numa revisão de literatura em engenharia de *software*. O método funciona da seguinte forma:

1. Primeiro passo ocorre a extração de *features* de texto.
2. O segundo passo é a amostra aleatória de um documento.
3. Terceiro passo é a amostragem de incerteza, a partir da construção de um modelo de classificação baseado no algoritmo SVM, é consultado os exemplos sem rótulo mais próximos do plano.
4. No quarto passo é feito um balanceamento de dados, o objetivo é ter o mesmo número de exemplos relevantes e irrelevantes. O algoritmo executa continuamente o treinamento, basta o usuário dar um novo *feedback* e o treino ocorre novamente.

Segundos experimentos reportados o *FASTREAD* exige de 20% a 50% menos estudos revisados pelo usuário chegando até 95% de recall.

Zhang et al. (2015) propuseram um MPV usando extração de texto e métricas de software, este método foi chamado de VULPRETICTOR. Para a classificação, foi usado um combinado de três modelos de aprendizagem sendo eles: *Random Forest*, *Naive Bayes* e *Decision Tree*. O conjunto de métricas e *features* de texto foram treinados separadamente gerando seis modelos de previsão. Em seguida foi construído um classificador chamado de *COMPOSER* que combina os seis classificadores e combina as saídas e

produz uma saída de confiança. O modelo foi aplicado em três bases de códigos diferentes. Os resultados obtidos foram: Para o *Drupal* 69% de *recall*, para os arquivos do *PHPMyAdmin* atingiu 33% de *recall* e para o *Moodle* 4% de *recall*.

O método de Yu et al. (2019), denominado HARMLESS, apresenta um MPV usando aprendizagem ativa e tem como objetivo minimizar erros e o esforço humano.

O HARMLESS funciona da seguinte forma:

1. **Primeiro passo**: Extração de *features*: é baseada em dois recursos de extração. Primeiramente, são utilizadas métricas de *software*, como o número de métodos declarados no arquivo de código-fonte, número de linhas de código do arquivo de código-fonte, entre outras. Em seguida, é realizado a extração de texto de código-fonte usando o método TD-IDF e usando apenas os 4000 *tokens* com melhor pontuação;
2. **Segundo passo**: Amostragem inicial aleatória na qual é selecionado arquivos que ainda não foram rotulados;
3. **Terceiro passo**: Inspeção de código, os arquivos selecionados no passo anterior passam por um revisor no qual são rotulados;
4. **Quarto passo**: Os arquivos rotulados passam uma por técnica de balanceamento de dados para ter o mesmo número de arquivos vulneráveis e não vulneráveis. Em seguida é feito o treinamento numa máquina de vetores de suporte (SVM);
5. **Quinto passo**: Previsão de erro, é aplicado um treinamento SVM nos arquivos rotulados apenas uma única vez, em seguida os arquivos que tem maior probabilidade são adicionados numa fila para inspeção redundante;
6. **Sexto passo**: Inspeção redundante, é solicitado o rótulo dos arquivos adicionados na fila no passo anterior;
7. **Sétimo passo**: Estimação de *recall* é designado rótulos temporários para arquivos não rotulados, após isso é calculado o *recall*. Caso atinja o *recall* desejado para o método, se não vá para o próximo passo;
8. **Oitavo passo**: Aplica-se duas estratégias de consulta. Primeiro a estratégia de incerteza, que consiste em pegar os exemplos mais próximos do hiperplano gerado pelo SVM. Após é aplicado a amostragem de certeza, consiste em selecionar os exemplos mais distantes do hiperplano. Os exemplos que foram selecionados pelas estratégias são adicionados numa fila Q e esses exemplos irão para o passo 3.

Os resultados do método são apresentados na Tabela 1. Este experimento foi realizado com o objetivo de parar no *recall* alvo, as *features* híbridas que são um combinando de métricas de *software* e recursos de texto não conseguiram chegar ao *recall* alvo. O custo é definido pela porcentagem de documentos analisados. Analisando a Tabela 1 pode-se concluir que as melhores *features* para este método foram as de ”Texto”.

4. Experimentos

Nesta seção serão descritos os experimentos realizados com o objetivo de avaliar como o método *knee* (projetado para diminuir esforço nas revisões sistemáticas da literatura) se comporta no contexto de um modelo de previsão de vulnerabilidade. Primeiramente será descrita a base de dados utilizada, em seguida a metodologia usada e, por último, apresentados os resultados.

Tabela 1. Resultado Harmless

Features	Recall	Custo
Texto	90%	15%
	95%	21%
	99%	43%
Híbrido	67%	8%
	70%	8%
	99%	46%

4.1. Base de Dados

Os dados usados são coletados do projeto *Mozilla Firefox*[1]. Este conjunto de dados foi criado revisando e marcando manualmente os defeitos de código do projeto *Mozilla Firefox* a partir de 21 de novembro de 2017. Este conjunto de dados[2] contém 28.750 arquivos de código-fonte do *Mozilla Firefox* nas linguagens C e C++. Destes arquivos, 271 apresentam algum tipo de defeito. Além dos arquivos de código-fonte, contém métricas de *software*.

4.2. Extração de *features*

Os dados de código-fonte encontram-se no formato textual, para conseguir utilizá-los num modelo de aprendizado de máquina é necessário estruturá-las. Para isso, foi usado o método TF-IDF com duas combinações de hiperparâmetros. A primeira combinação, baseada no trabalho de Li and Kanoulas (2020), utilizou o hiperparâmetro *stop_words=english* que consiste em eliminar palavras que costumam aparecer muito nos textos em inglês, e não são relevantes para a aprendizagem como as palavras: *the*, *and* e *him*. A segunda combinação de hiperparâmetros foi a usada no trabalho de Yu et al. (2019), os hiperparâmetros usados são: *norm=l2* normaliza os resultados, e *max_features=4000* que consiste em selecionar apenas os 4000 *tokens* melhores pontuados.

As métricas de *software* foram extraídas a partir da ferramenta *SciTools' Understand* [3]. Tais métricas são descritas a seguir:

1. *CountClassBase*: O número de subclasses no arquivo de código-fonte;
2. *CountClassCoupled*: O acoplamento das classes no arquivo de código-fonte;
3. *CountClassDerived*: O número de subclasses derivadas de classes originadas no arquivo de código-fonte;
4. *CountDeclInstanceVariablePrivate*: O número de variáveis privadas declaradas no arquivo de código-fonte;
5. *CountDeclMethod*: O número de métodos declarados no arquivo de código-fonte;
6. *CountInput* : O número de chamadas externas feitas para este arquivo de código-fonte;
7. *CountOutput*: O número de chamadas externas feitas a partir deste arquivo de código-fonte;
8. *Cyclomatic*: A Complexidade Ciclomática deste arquivo de código-fonte;

[1]https://www.mozilla.org/pt-BR/firefox/new/

[2]https://github.com/ai-se/Mozilla_Firefox_Vulnerability_Data

[3]https://www.scitools.com/

9. *CountLine*: O número de linhas de código (excluindo comentários e espaços em branco) neste arquivo de código-fonte;
10. *MaxInheritanceTree*: O tamanho da folha máxima na árvore de herança que sai deste arquivo de código-fonte;
11. *Crashes*: O número de vezes que o arquivo teve uma falha.

As *features* de métricas de *software* foram normalizadas usando a norma l2. As *features* de texto e de métricas de *software* foram combinadas, esta combinação foi chamada de híbrida.

4.3. Métricas de avaliação

No intuito de validar o desempenho do experimento proposto, são usados duas métricas de avaliação: *recall* e o custo. A métrica de avaliação *recall* é a principal métrica de avaliação para tarefas de recuperação de informação, ela calcula a relação de documentos relevantes encontrados e o total de documentos relevantes [Cruz 2019]. Neste caso, os documentos relevantes serão os arquivos de código-fonte que contém algum tipo de defeito. O *recall* pode ser expressado matematicamente pela fórmula Equação 5. Na qual R é o *recall*, RAD é o número arquivos com defeitos encontrados e TAD é o número total de arquivos com defeitos.

$$R = \frac{RAD}{TAD} \tag{5}$$

Já custo é medido pela porcentagem de arquivos revisados, expressado pela fórmula Equação 6. Na qual C se refere ao custo, AR é o número de arquivos revisados e TA é o número total de arquivos.

$$C = \frac{AR}{TA} \tag{6}$$

4.4. Configuração do experimento

Os experimentos foram realizados usando o método knee[4]. Com objetivo de avaliar como uma ferramenta originalmente projetada para revisão de literatura, se comporta quando aplicada como um modelo de previsão de vulnerabilidade, e analisar quais *features* e configurações produzem o melhor resultado.

O experimento foi executado com diferentes combinações de configurações, *features* e tópicos. Os algoritmos de aprendizagem de máquina usados nos experimentos foram Máquina de Vetores de Suporte (SVM) e a regressão logística. A SVM foi testado com duas configurações de hiperparâmetros. A primeira combinação de hiperparâmetros (chamada de A), é original do método *knee*, e define que: *gamma='scale'*. Já a combinação de hiperparâmetros (chamada de B), foi gerada pelo *GridSearchCV* que escolhe a melhor combinação a partir de um teste com todas as combinações. A melhor combinação resultante de B foi: *C=10000, gamma=0.001, kernel='linear'*.

O método *Knee* apresenta dois parâmetros: ρ e β. O ρ é um parâmetro de parada, que define em que ponto a curvatura de aprendizagem converge e é possível parar

[4]https://github.com/dli1/auto-stop-tar

o método. Quando definido um ρ menor que 6, estes não alcançaram um *recall* mínimo (menos de 80%). Já para valores de ρ maiores que 10 foi apresentado um custo alto de rotulação (cerca de 50% da base). O ρ que obteve os resultados mais promissores foi o ρ = 10. Por este motivo os experimentos apresentados nesta seção usam um valor de ρ = 10.

O β representa o tamanho da mínimo da amostra para convergência do método, original do método tem um valor de 100. Utilizando o valor de β=100, para alguns experimentos *recall* não chegava à 3%. Desta forma, experimentalmente foi avaliado que o valor de β = 1000 obteve um bom resultado.

O tópico usado no proposito original é uma consulta (*query*) relacionada a algum arquivo. Os tópicos usados neste experimento é um arquivo código-fonte que contenha algum tipo de defeito. No qual, o objetivo é encontrar documentos semelhantes a este.

4.5. Análise dos experimentos

Os resultados usando a técnica de aprendizagem SVM são apresentados na Tabela 2. A primeira coluna refere-se como as *features* foram geradas. "Texto" refere-se as *features* extraídas dos arquivos de código-fonte, "Métricas" se referem as métricas de *software* relatadas na seção 4.2, "Híbridas" refere-se a junção das "Métricas" e "Texto", e "PCA"se refere a redução de dimensionalidade usando o método PCA (Análise de componentes principais). A coluna "Custo", refere-se ao custo de rotulação. A coluna "N *features*" refere-se ao número de características utilizadas em cada experimento. Os "Hiperparâmetros do SVM" se referem-se a combinação de hiperparâmetros relatados na seção 4.4.

Pode-se visualizar que as *features* "Métricas" não apresentaram um bom resultado. Já as *features* "Híbridas" e "Texto" usando "4000 N *features*" e os Hiperparâmetros SVM "A" considerando o desvio padrão tiveram um empate. Também pode-se ver que os hiperâmetros SVM "A" apresentam um melhor resultado se comparado ao "B". O experimento usando o PCA não obteve um bom resultado, uma vez que o método somente convergiu após rotular a base inteira. As configurações mais promissoras encontrada nos experimentos foram de *features* "Texto"e "Hibridas". Além disso, observou-se que utilizar todas as *features* (419572) pode gerar impactos positivos na redução do custo do processo.

Tabela 2. Recall vs Custo em diferentes configurações usando SVM

Features	Recall	Custo	N features	Hiperparâmetros SVM
Texto	97,04% $\pm$ 0,5	34% $\pm$ 3	4000	A
	96,12% $\pm$ 1,06	36% $\pm$ 4	4000	B
	95,02% $\pm$ 1,36	27% $\pm$ 2	419572	B
Híbridas	96,12% $\pm$ 0,51	31% $\pm$ 2	4011	A
	94,83% $\pm$ 1,22	36% $\pm$ 14	4011	B
Métricas	73,98% $\pm$ 15,67	33% $\pm$ 16	11	A
	64,33% $\pm$ 16,81	61% $\pm$ 26	11	B
PCA	100%	100%	100	A

Os resultados usando a técnica de aprendizagem regressão logística são apresentados na Tabela 3. Nesta tabela, os resultados são similares a Tabela 2. Pode-se visualizar

Tabela 3. Recall vs Custo em diferentes configurações usando Regressão Logística

Features	Recall	Custo	N features
Texto	96,67% ± 0,65	24% ± 2	4000
	97,04% ± 0,15	22% ± 1	419572
Híbridas	97,41% ± 0,38	24% ± 2	4011
Métricas	99,26% ± 0,49	48% ± 8	11

que todos os experimentos da Tabela 3 apresentam resultados mais promissores que os da Tabela 2. As *features* de "Métricas" também não obtiveram um bom resultado neste experimento, já as de "Texto" e "Híbridas"obtiveram resultados similares e promissores.

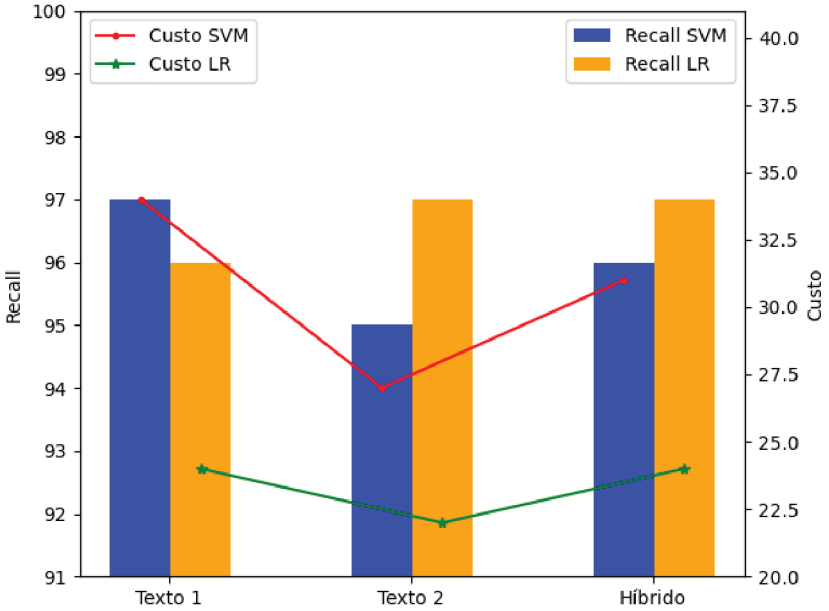

Figura 4. Relação SVM x Regressão Logística

No gráfico apresentado na Figura 4 é ilustrado a relação entre custo e *recall* comparando a SVM e a Regressão Logística. São reportadas as *features* de "Texto"e as "Híbridas". O "Texto 1" refere-se as *features* com tamanho de 4000, o "Texto 2" refere-se as *features* com tamanho de 419572, e o "Híbrido" refere-se as *features* "Híbridas". As *features* de "Métricas" e "PCA", e Hiperparâmetros SVM "B"foram desconsideradas neste gráfico por não apresentar um resultado promissor.

Pode-se visualizar que a linha de "Custo LR" (eixo y direito) representando os experimentos treinados com a regressão logística fica abaixo da linha "Custo do SVM", o que representa que usar o algoritmo de aprendizagem de máquina Regressão Logística, é melhor do que o SVM (para este experimento). Também pode-se notar que o experimento o "Texto 2" (eixo x) treinado com a regressão logística que demonstra o segundo experimento da Tabela 3, no gráfico este teve uma leve queda na linha de custo em relação aos demais também treinados com a regressão logística, porém não pode-se dizer que essa forma de extração de *features* é melhor que as demais, pois no gráfico não está sendo

considerado o desvio padrão.

Este experimento demonstrou que o método *knee* [Li and Kanoulas 2020] pode ser aplicado no contexto de um modelo de previsão de vulnerabilidade (MPV). Os resultados usando *features* "Texto" apresentados na Tabela 3 são semelhantes aos resultados do método *HARMLESS* apresentados na Tabela 1. Os resultados usando as *features* "Híbridas" obtiveram um resultado mais promissor no método *knee* comparado aos resultados do método *HARMLESS*.

Para a construção das *features* "Híbridas" é necessário realizar a extração de métricas de *software* antes de passar para um MPV, já a extração de "Texto" é feita diretamente no MPV. No experimento Tabela 3 a diferença no resultado entre as *features* de "Texto"e "Híbridas" é muito pequena, ou seja, a extração de *features* de métricas de *software* para junta-lá com as de "Texto" e formar as "Híbridas"é inviável dado que com apenas as *features* de "Texto"alcançam um resultado similar.

5. Conclusão

Esse artigo teve como objetivo analisar experimentalmente quais *features* e configurações causam um melhor impacto no método *knee*, no contexto de um modelo de previsão de vulnerabilidade.

A experimentação demostrou que o método *knee* pode ser usado também como modelo de previsão de vulnerabilidade. Também descartou o uso de métricas de *software*, pois não causaram impacto no método, e tem um alto custo de extração. Além disso, foi demonstrado que o método pode encontrar cerca de 97% das vulnerabilidades, rotulando cerca de 22% dos arquivos. Em alguns experimentos o *knee* se demonstrou superior à um método projetado especificamente para identificar vulnerabilidades.

O método *knee* propõe que o revisor não comete erros, ou seja, se o revisor atribuí um rótulo errado, pode comprometer todos os rótulos já atribuídos. Nos próximos trabalho pretende-se utilizar outras bases de dados para o mesmo experimento e também realizar um tratamento de erros de rotulação.

Referências

Aranha, C. and Passos, E. (2006). A tecnologia de mineração de textos. *Revista Eletronica de Sistemas de Informação*, 5(2).

Christian, H., Agus, M. P., and Suhartono, D. (2016). Single document automatic text summarization using term frequency-inverse document frequency (tf-idf). *ComTech: Computer, Mathematics and Engineering Applications*, 7(4):285–294.

Cleverdon, C. W. (1974). User evaluation of information retrieval systems. *Journal of documentation*, 30(2):170–180.

Corcovia, L. O. and Alves, R. S. (2019). Aprendizagem de mÁquina e mineraÇÃo de dados: avaliação de métodos de aprendizagem. *Revista Interface Tecnológica*, 16(1):90–101.

Cruz, L. A. (2019). Modelo para recuperação de informação em repositórios institucionais utilizando a técnica de sumarização a partir da seleção de atributos do cassiopeia.

Dreiseitl, S. and Ohno-Machado, L. (2002). Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359.

Duarte, D. and Ståhl, N. (2019). *Machine Learning: A Concise Overview*, pages 27–58. Springer International Publishing, Cham.

Li, D. and Kanoulas, E. (2020). When to stop reviewing in technology-assisted reviews: Sampling from an adaptive distribution to estimate residual relevant documents. *ACM Transactions on Information Systems (TOIS)*, 38(4):1–36.

Li, T., Mei, T., Kweon, I.-S., and Hua, X.-S. (2010). Contextual bag-of-words for visual categorization. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(4):381–392.

Mangasarian, O. L. and Musicant, D. R. (2000). Active support vector machine classification. In *NIPS*, number 7. Citeseer.

Morais, C., Meirelles, P., and Morais, E. (2012). Kalibro metrics: um serviço para monitoramento e interpretaç ao de métricas de código-fonte.

Pfleeger, S. L. (2004). *Engenharia de software: teoria e prática*. Prentice Hall, São Paulo.

Satopaa, V., Albrecht, J., Irwin, D., and Raghavan, B. (2011). Finding a"kneedle"in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE.

Settles, B. (2009). Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Shamal, P., Rahamathulla, K., and Akbar, A. (2017). A study on software vulnerability prediction model. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 703–706. IEEE.

Yu, Z., Kraft, N. A., and Menzies, T. (2018). Finding better active learners for faster literature reviews. *Empirical Software Engineering*, 23(6):3161–3186.

Yu, Z. and Menzies, T. (2018). Total recall, language processing, and software engineering. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, pages 10–13. ACM.

Yu, Z., Theisen, C., Williams, L., and Menzies, T. (2019). Improving vulnerability inspection efficiency using active learning. *IEEE Transactions on Software Engineering*.