Um estudo sobre reutilização de treinamento em Modelos de Previsão de Vulnerabilidade

Matheus Vinícius Todescato, Guilherme Dal Bianco[1]

[1]Universidade Federal da Fronteira Sul
Campus Chapecó
Chapecó – SC – Brazil

mvtodescato@hotmail.com, guilherme.dalbianco@uffs.edu.br

Abstract. *Finding bugs or code failures in systems can be a complex and costly task. An alternative to reduce user effort is to apply the Vulnerability Prediction Model (VPM). A VPM uses machine learning techniques to identify code parts with possible bugs. For this, VPM needs training (source code files containing bugs) to build a prediction model. Such a problem is known as cold-start, in which the method has no information to start the bugs identification process. In this work, the objective is to experimentally evaluate the reuse of training between projects to reduce the manual cost of the process when we aim to identify all (or almost) bug codes.*

Resumo. *Encontrar bugs ou falhas de código em sistemas pode ser uma tarefa extremamente complexa e onerosa. Uma alternativa para diminuir o esforço do usuário é aplicar o Modelo de Previsão de Vulnerabilidade (MPV). Um MPV utiliza técnicas de classificação e aprendizagem ativa para identificar trechos de código com possíveis bugs. Para isso, o MPV depende de um treinamento inicial (arquivos de código contendo bugs) na construção de um modelo de predição. Tal problema, conhecido como partida fria ou cold-start, surge quando o método não tem exemplos representativos para o início do processo. Neste trabalho, o objetivo é avaliar experimentalmente a reutilização de treinamento entre projetos com intuito de aliviar o impacto da partida fria quando se deseja encontrar todos (ou quase todos) arquivos de bug.*

1. Introdução

Com o avanço da tecnologia, os códigos-fonte de *softwares* têm ficado cada vez mais complexos. Por exemplo, o conjunto do *Mozilla Firefox*[1] é formado por mais de 28 mil arquivos de código-fonte. Analisar em busca de *bugs* um conjunto desse porte manualmente representa uma tarefa inviável e tediosa. Uma alternativa, é utilizar Modelos de Previsão de Vulnerabilidade (MPVs). Os MPVs tem como objetivo detectar arquivos de código-fonte que contenham alguma vulnerabilidade. Os MPVs são baseados em aprendizagem de máquina e o treinamento é feito a partir de erros conhecidos. Sabendo qual arquivo é vulnerável, o desenvolvedor pode focar na resolução do possível erro [Yu et al. 2018].

A tarefa de encontrar todas e somente as vulnerabilidades pertence à classe de problemas de recuperação de informação denominada *revocação total* (ou *high recall*). O

[1]https://bugzilla.mozilla.org/home

objetivo da revocação total é otimizar o custo na obtenção de altos níveis de revocação (idealmente em torno de 100%) com um humano assessor no ciclo [Yu et al. 2019]. Atingir uma alta revocação é importante para reduzir vulnerabilidades que podem causar impactos significativos quando liberadas aos usuários.

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]. Como, 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 mitigar o esforço do revisor humano é aplicar abordagens de aprendizagem ativa [Settles 2009a]. A aprendizagem ativa tem como objetivo reduzir o esforço do usuário na rotulação, identificando os documentos mais informativos para o processo de aprendizagem [Settles 2009a]. Por exemplo, o método [Cormack and Grossman 2016a], chamado de KNEE, explora aprendizagem ativa para determinar quais documentos devem ser rotulados pelo usuário no contexto de textos livres (sem estrutura). Além de diminuir o esforço de rotulação, o método almeja estimar o número de documentos relevantes da coleção, podendo interromper, com segurança, o método antes de analisar todos os documentos.

Um dos desafios que persiste ao adotar a aprendizagem ativa é a geração do conjunto de treinamento inicial (semente). Este conjunto, utilizado pelo algoritmo de aprendizado no início do processo, é primordial para a identificação de padrões que configuram um documento relevante [Settles 2009a]. Tal etapa possui grande impacto no esforço despendido pelo usuário, posto que pode acelerar ou atrasar o aprendizado do algoritmo. Ou seja, se nenhum trecho com *bug* é adicionado ao treinamento, o algoritmo de ranqueamento dificilmente conseguirá produzir um bom ordenamento. Tal problema é conhecido como *cold-start* [Fisichella et al. 2009], estando presente não apenas no campo da aprendizagem ativa mas também em algoritmos de aprendizado em geral.

Este trabalho tem como objetivo avaliar a reutilização de treino para MPVs com a intenção de aprimorar a qualidade do processo no contexto da aprendizagem ativa. A hipótese é que na ausência de trechos marcados como *bugs*, pode-se reutilizar códigos rotulados (treinamento) de outros projetos para acelerar o processo de aprendizado do método ativo, reduzindo assim o esforço de rotulação. Nesse contexto, este artigo busca responder às seguintes questões de pesquisa:

- A reutilização do treinamento de outras bases de dados pode auxiliar no problema da *partida fria* do método?
- O número de arquivos com *bugs* fornecidos para o método ativo impacta no ganho de informação do modelo ativo?

Foram conduzidos experimentos em conjunto de código reais com objetivo de responder às questões de pesquisa. Dessa forma, foi observado que reutilizando um conjunto controlado de arquivos pode-se obter resultados promissores em relação à revocação e ao custo de rotulação do processo.

O texto a seguir está organizado da seguinte forma. Na Seção 2 são apresentados os conceitos básicos deste trabalho. Seção 3, os trabalhos relacionados são descritos. Os experimentos são apresentados na Seção 4. Por fim, a conclusão é descrita.

2. Referencial Teórico

Nesta seção, são apresentados os principais conceitos e métodos importantes para a compreensão deste trabalho.

2.1. Extração de Características

Documentos são tradicionalmente apresentados em formato de texto livre. Tal representação deve ser modificada, geralmente para o formato numérico, a fim de possibilitar seu processamento por algoritmos de aprendizagem de máquina. Uma das técnicas básicas existentes com este propósito é o BoW (sigla para *Bag of Words*), que consiste em contabilizar a ocorrência de palavras em um dado documento [Manning et al. 2018]. Desta forma, o documento passa a ser representado por um vetor contendo o número de ocorrências de cada palavra que o compõe.

Alguns documentos podem apresentar tamanho elevado em comparação a outros em uma mesma base de dados e, consequentemente, o número de ocorrências de suas palavras será maior. Neste cenário, a utilização do BoW fica comprometida, uma vez que sua representação torna-se tendenciosa para os documentos mais extensos. Técnicas mais sofisticadas para extração de características não possuem tal inconveniente. É o caso do TF-IDF (*Term Frequency - Inverse Document Frequency*) que consiste em um algoritmo de extração de características que, semelhante ao BoW, utiliza a frequência das palavras para construir os vetores de característica. Porém, o TF-IDF considera também a frequência dos termos perante à base como um todo, fazendo com que o resultado seja independente do tamanho dos documentos.

Para o funcionamento esperado dos MPV é necessário a extração de características informativas do código fonte. Para isso, três técnicas podem ser utilizadas: *mineração de texto*, *métricas de software* ou combinando ambos. *Métricas de software* expressam características numéricas sobre o código-fonte. Algumas das métricas mais usadas são referente ao tamanho do código e a sua complexidade [Morais et al. 2012]. Já o modelo baseado em *mineração de texto* explora o uso de técnicas de extração de *tokens* no código (por exemplo, o TF-IDF). A ideia é extrair os pesos dos *tokens* de acordo com sua frequência [Shamal et al. 2017].

2.2. Aprendizado Supervisionado

Algoritmos de aprendizado supervisionado tem como propósito fornecer uma função que mapeia dados para determinadas classes [Manning et al. 2018]. Para isso, é necessário que um conjunto de treinamento (conjunto de dados compostos por rótulos) suficientemente informativo seja fornecido para o aprendizado de padrões existentes.

Algoritmos supervisionados podem ser divididos em classificadores e regressores [Goodfellow and Courville 2016]. O primeiro tem como resultado um valor discreto, representando uma classe à qual o exemplo de entrada pertence. O segundo resultará em um valor contínuo, que representa uma possível resposta (saída) de acordo com a entrada. Entre os algoritmos baseados em técnicas supervisionadas pode-se indicar [Goodfellow and Courville 2016]: *Naïve Bayes*, *Support Vector Machines* - SVM e Redes Neurais Artificiais (ANN, do inglês *Artificial Neural Networks*).

Naïve Bayes é um algoritmo probabilístico, cuja fundamentação remete ao teorema de Bayes, por meio do qual é possível obter probabilidades condicionadas. Seu

aprendizado é efetuado buscando quais termos no documento fornecem mais evidências de pertencer à determinada classe [Manning et al. 2018]. Para a aplicação do algoritmo, assume-se uma forte independência entre as características analisadas. Já o algoritmo SVM visa dividir o conjunto de dados em segmentos. Esta divisão é feita encontrando um hiperplano de tal modo que, a distância entre o hiperplano e os pontos mais próximos a ele seja máxima [Manning et al. 2018]. Embora o algoritmo seja primordialmente para classificação binária (onde há apenas duas classes), é possível utilizá-lo com mais classes.

2.3. Aprendizado não Supervisionado

Algoritmos não supervisionados não requerem dados rotulados para o treinamento do método. Nesta categoria, instâncias x_i pertencentes a um conjunto de dados X não estão atrelados à um rótulo y, e por consequência, a classificação (ou regressão) dá-se por meio de inferências acerca de padrões encontrados nos dados [Goodfellow and Courville 2016]. Alguns algoritmos de clusterização são exemplos do emprego de aprendizado não supervisionado.

A clusterização tem como finalidade dividir o conjunto de dados em grupos (*clusters*) de modo que os exemplos dentro de um mesmo grupo sejam semelhantes entre si [Goodfellow and Courville 2016]. O algoritmo *K-means*, por exemplo, promove a geração de grupos, computando iterativamente as distâncias (Euclidiana, Manhattan, dentre outras) das instâncias para o seu centroide [Manning et al. 2018]. O centroide de um *cluster* é definido como a média das distâncias - tomadas sobre os valores das *features* - de todos os documentos contidos nele.

2.4. Aprendizado Ativo

Em muitas aplicações, a utilização de técnicas de aprendizado de máquina - em essência, na modalidade supervisionada - requer um alto grau de qualidade dos dados de treinamento sobre os quais o algoritmo irá operar [Settles 2009a]. Sobretudo, a existência de dados rotulados é imprescindível. Não obstante, há cenários, como nos MPVs, em que a obtenção de rótulos representa um custo elevado, inviabilizando a utilização de técnicas supervisionadas. Assim sendo, o emprego de técnicas alternativas mostra-se de fundamental importância, dentre as quais há o aprendizado ativo.

Técnicas de aprendizado ativo partem do pressuposto de que o algoritmo irá escolher quais documentos serão rotulados, evitando inserção de instâncias não informativas junto ao treinamento [Settles 2009b]. De maneira sucinta, o algoritmo selecionará amostras informativas de um conjunto de dados cuja rotulação é desconhecida, e irá solicitar para que um usuário defina o rótulo. Em seguida, tais instâncias passam a integrar o conjunto de treino (com rótulos) que será consumido por um algoritmo supervisionado.

Existem várias abordagens para a seleção das instâncias que serão rotuladas ativamente pelo revisor, como a incerta e a convicta [Yu et al. 2018]. Na primeira, são selecionadas as instâncias mais próximas da fronteira de decisão, ou seja, as instâncias em que o modelo tem mais "dúvidas". Na segunda estratégia são selecionadas as instâncias mais distantes da fronteira de decisão. Por exemplo, considerando a utilização do modelo SVM, durante a amostragem incerta serão escolhidas as instâncias mais próximas do hiperplano, enquanto na convicta serão escolhidas as instâncias mais distantes do hiperplano.

3. MPVs e Trabalhos Relacionados

Os Modelos de Previsão de Vulnerabilidades (MPVs) tem como objetivo encontrar a maior parte das vulnerabilidades em códigos fontes (alto *recall*) com esforço humano reduzido. No entanto, os MPVs possuem alguns desafios [Yu et al. 2018], como por exemplo: (*i*) a construção do treinamento inicial, já que projetos em versões iniciais podem não conter informações iniciais; (*ii*) definição, pelo usuário, da *recall* (percentual de vulnerabilidades encontradas) desejado ao fim da inspeção; e (*iii*) mecanismos de correção de falha humana (revisão incorreta). O problema da partida fria (*i*), alvo deste trabalho, pode resultar no atraso do aprendizado do método, uma vez que a ausência de exemplos de código com *bugs* reduz a capacidade da correta identificação de arquivos relevantes (*bugs*).

No MPV proposto em [Zhang et al. 2015], é explorando o uso de um comitê de classificadores para identificação de arquivos com *bug*, chamado de VULPRETICTOR. O método combina três modelos de aprendizagem (*Random Forest*, *Naive Bayes* e *Decision Tree*). Conjuntos de dados são gerados combinando *features* de texto e métricas de software para a criação de modelos de classificação. Por fim, a saída dos classificadores é combinada para medir o grau de confiança. O método não explora a geração do treinamento inicial, alvo deste trabalho.

O método proposto em [Yu et al. 2019], denominado HARMLESS, apresenta um MPV utilizando aprendizagem ativa com o objetivo de encontrar todos os arquivos com *bugs* com reduzido esforço do usuário. O HARMLESS explora a aprendizagem ativa para incrementalmente selecionar os arquivos com maior probabilidade de representarem *bugs*. Para isso, são selecionados os arquivos mais promissores a conterem *bugs* segundo o classificador, criando-se assim um ranqueamento. Ativamente, o usuário é requisitado para rotular os arquivos no topo do ranque, permitindo incrementar o treinamento. Diversas interação são requisitadas entre o usuário e o HARMLESS até atingir o *recall* desejado (estimado pelo método). Foram utilizadas duas abordagens para a criação do treinamento inicial nesse processo, (*i*) amostragem aleatória, onde arquivos de código fonte são escolhidos aleatoriamente para serem analisados, e (*ii*) amostragem com conhecimento, que considera a existência de características informativas sobre vulnerabilidades.

Já no método KNEE é proposto uma abordagem ativa com convergência automática no contexto de uma alta revocação em textos livres (base de dados envolvendo emails, artigo científicos, entre outros) [Cormack and Grossman 2016a]. Mais especificamente, é utilizado um método guloso para identificar o ponto de estabilização da curva de revocação. O ponto de parada é baseado na noção dos pontos de curvatura máxima em um conjunto de dados. A amostragem inicial é construída sinteticamente a partir da consulta do usuário. O método se mostrou bastante efetivo no contexto de texto livre , no entanto, não foi avaliado no cenário de MPVs.

No trabalho [Zhang et al. 2020] é explorado o uso de MPVs reutilizando diferentes versões do sistema para treinamento. O uso de versões históricas permite obter informações referentes a *bugs* e dos padrões de trechos de *bugs*. Nesse sentido, é proposto uma abordagem baseada em clusterização para selecionar trechos de código informativos de versões históricas para o treinamento do classificador. No entanto, nas primeiras versões do sistema, tais versões históricas não estão disponíveis, o que inviabiliza sua aplicação.

Diferentemente dos métodos apresentados acima, este trabalho tem como objetivo analisar se é possível auxiliar o processo de aprendizagem ativa a partir da reutilização de arquivos de outros projetos, já rotulados. Nesse contexto, será explorado o uso do método ativo KNEE, voltado para textos livres, no contexto da predição de arquivos com *bugs*.

4. Experimentação

Nesta seção, serão descritos os experimentos que tiveram como objetivo avaliar o impacto da reutilização de treinamento em diferentes bases de dados (projetos) sobre a revocação e o custo. Além disso, poderá ser observado como o método KNEE originalmente projetado para alta revocação em textos livres, se comporta quando aplicada no contexto de um MPV. Primeiramente, serão descritas as bases de dados utilizadas, em seguida as métricas usadas, configuração dos experimentos e, por último, será apresentado a análise dos resultados.

4.1. Base de Dados

As bases de dados utilizadas neste trabalho são públicas e estão disponíveis em [Tóth et al. 2016]. O conjunto de dados foi construído com objetivo de contribuir para aplicações de predição de *bugs*. As bases envolvem códigos utilizando a linguagem de programação Java na plataforma GitHub[2]. Foram selecionadas 105 versões de lançamentos de projetos ao longo de seis meses para a construção do conjunto de dados. Utilizando técnicas de extração de características foram gerados mais de 183 mil arquivos de código.

A Tabela 1 descreve com detalhes cada uma das bases de dados utilizadas. A coluna "Base de dados" contém os nomes de cada base de dados, a coluna "# Arq. de código" contém o número de arquivos de códigos e, por fim, a coluna "# Arq. de Bug" apresenta a quantidade de arquivos que contém *bugs*. A coluna "Prevalência" demonstra a porcentagem de arquivos com *bugs* em relação ao total.

Tabela 1. Descrição da bases de dados envolvendo os arquivos de código fonte.

Base de dados	# Arq. de código	# Arq. de *Bug*	Prevalência
mct	6091	23	0.004
junit	5432	87	0.016
anttlr4	2353	53	0.023
elasticsearch	54562	3474	0.064
android	639	48	0.075

4.2. Métricas de avaliação

No intuito de validar o desempenho do experimento proposto, são usados duas métricas de avaliação: revocação e o *Custo*. A métrica de avaliação revocação ou *recall* calcula a relação dos documentos relevantes encontrados e o total de relevantes [Cruz 2019]. Neste trabalho, os documentos relevantes serão os arquivos de código-fonte que contém algum tipo de defeito. O *recall* pode ser expresso matematicamente pela Equação 1. Na qual, RAD representa o número arquivos com defeitos encontrados e TAD representa o número total de arquivos com defeitos.

[2]www.github.com

$$Revocação(Recall) = \frac{RAD}{TAD} \quad (1)$$

Já o *Custo* é medido pela porcentagem de arquivos revisados, definido pela Equação 2. AR representa número de arquivos revisados e TA representa número total de arquivos.

$$Custo = \frac{AR}{TA} \quad (2)$$

Para auxiliar na comparação direta das execuções, será reportado a curva (ou a área) envolvendo *recall* vs. Custo. Quanto maior a área abaixo da curva, maior será o *recall* e menor será o custo.

4.3. Configuração do experimento

Os experimentos foram realizados utilizando a implementação do método KNEE disponibilizado como código aberto [3]. O algoritmo de aprendizagem de máquina utilizado foi o SVM, utilizando a parametrização proposta em [Li and Kanoulas 2020]. Originalmente, o método KNEE conta com uma entrada textual do usuário (*query*), por exemplo, a *string* inserida pelo usuário no buscador. Já no contexto do MPVs, na qual não está disponível a *string* de consulta, serão executados as seguintes configurações: *(i)* "original" onde será fornecido um arquivo com *bugs* selecionado randomicamente na mesma base de dados e rotulado pelo usuário para início do processo de aprendizagem; *(ii)* "1 arq. treino" na qual será utilizado um arquivo de treino de outra base de dados para início do processo; *(iii)* "5 arq. treino", na qual, serão extraídos cinco arquivos de treino de outra base de dados; *(iv)* "10 arq. treino", serão utilizado 10 arquivos de treino de outra base de dados; e por fim, *(v)* "100% arq. treino" será utilizado, como ponto de partida, todo conjunto já rotulado de outra base de dados.

Os conjuntos de dados "anttlr4", "mct" e "junit" foram utilizados para compor os testes. Já as bases de dados "android" e "elasticsearch" foram utilizados para compor os exemplos já rotulados de treinamento por representar as distribuições com menor e maior número de arquivos fonte, respectivamente.

4.4. Análise dos Experimentos

As Tabelas 2, 3 e 4 possibilitam comparar a área de cada curva de *Recall* vs *Custo*. Como pode-se perceber, ao adicionar ao treinamento inicial 1 único arquivo com *bug* ("1 arq. treino" de outra base de dados não foi produzido ganho de aprendizado ou redução no custo de rotulação quando comparado às demais áreas de curvatura. Já, ao incrementar para 5 ou 10 arquivos com *bug*, foi possível obter os melhores resultados em comparação com os demais em todas as bases de dados. Por exemplo, a base de dados "anttlr4" obteve um ganho de 11% na área da curva se comparado ao método KNEE original.

Ao adicionar toda a base de dados (ou seja, o processo de aprendizagem inicia com todos os arquivos da outra base de dados já rotulada) foi obtido um baixo valor de área se comparado aos demais. Acredita-se que esse comportamento seja devido a dominância de exemplos da base de dados reutilizada, isso dificulta a adaptação do modelo ativo de

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

predição para o padrão de *bugs* presente na base de dados alvo. Por exemplo, na Tabela 3 a área da curva quanto utilizado o como treinamento inicial 100% da base de dados *android* resultou em uma área 15% inferior ao método "original", claramente a inserção do treinamento prejudicou o aprendizado do método ativo KNEE.

Os gráficos 1, 2 e 3 detalham as curvas envolvendo o *recall* vs *custo*. Conforme já dito, quando mais elevada for a curva, maior será o *recall* e menor será o custo de rotulação. A linha preta representa a execução *Original* do método KNEE. Tais gráficos auxiliam no entendimento do comportamento observado nas Tabelas 2, 3 e 4. Ao utilizar como treino a base de dados rotulada completa (*100% arq. treino*) podemos observar uma curva com baixa aceleração, resultado em um atraso no ganho do *recall*. Com a mesma configuração, a base de dados *mct* apresentou um comportamento divergente, já que sua curva mostrou uma melhora acentuada no *recall* nos pontos finais (em torno de 60% do *recall*). Pretende-se executar mais experimentos em trabalhos futuros para explicar tal comportamento. As curvas em "verde" e "amarelo" (*treino: 5 arq. treino e treino: 10 arq. treino*) mostram uma aceleração promissora no início do processo ativo na maior parte dos gráficos. Dessa forma, ao utilizar um conjunto de treino de 5 ou 10 arquivos, podemos observar as melhores curvas quando comparado aos demais.

Desta forma, estes experimentos demonstraram que utilizando alguns arquivos rotulados entre projetos se mostra promissor no processo de construção do treinamento inicial dos MPVs. Porém, a inserção de muitos exemplos podem impactar negativamente na capacidade de identificação de novos padrões pelo modelo de predição no método KNEE.

Tabela 2. Área da curva para a base de dados *Anttlr4*.

Base Treino	Base Teste	Tam. Treino	Área
android	anttlr4	1 arq. treino	70.75
android	**anttlr4**	**5 arq. treino**	**80.35**
android	anttlr4	10 arq. treino	75.00
android	anttlr4	100% arq. treino	68.33
elasticsearch	anttlr4	1 arq. treino	70.09
elasticsearch	anttlr4	5 arq. treino	65.75
elasticsearch	anttlr4	10 arq. treino	77.59
elasticsearch	anttlr4	100% arq. treino	53.24
original	anttlr4	-	72.31

Tabela 3. Área da curva para a base de dados *junit*.

Base Treino	Base Teste	Tam. Treino	Área
android	junit	1 arq. treino	108.19
android	**junit**	**5 arq. treino**	**108.51**
android	junit	10 arq. treino	108.37
android	junit	100% arq. treino	92.90
elasticsearch	junit	1 arq. treino	102.80
elasticsearch	junit	5 arq. treino	105.69
elasticsearch	junit	10 arq. treino	98.74
elasticsearch	junit	100% arq. treino	85.59
original	junit	-	105.13

Tabela 4. Área da curva para a base de dados *mct*.

Base Treino	Base Teste	Tam. Treino	Área
android	mct	1 arq. treino	71.93
android	mct	5 arq. treino	69.35
android	mct	10 arq. treino	68.48
android	mct	100% arq. treino	58.48
elasticsearch	mct	1 arq. treino	66.86
elasticsearch	mct	5 arq. treino	70.07
elasticsearch	**mct**	**10 arq. treino**	**77.17**
elasticsearch	mct	100% arq. treino	67.68
original	mct	-	57.52

5. Conclusão

Este artigo teve como objetivo analisar experimentalmente a reutilização de treinamento no contexto da previsão de vulnerabilidades, ou seja, na identificação de *bugs* em códigos.

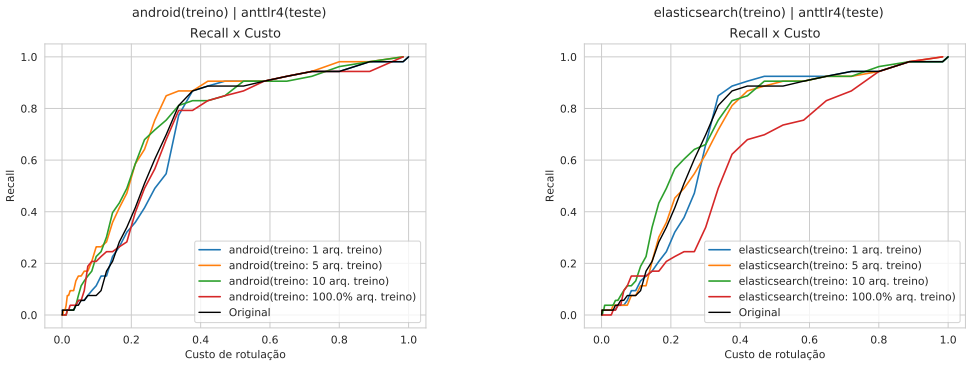

Figura 1. Curva aplicando a base de dados *anttl4*.

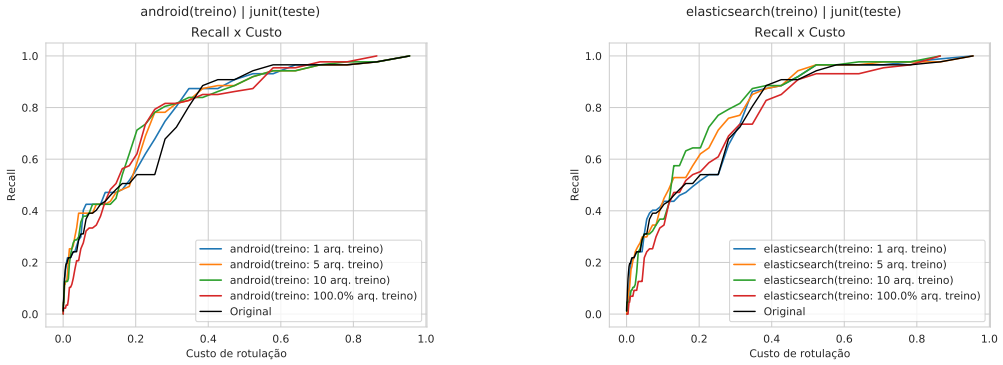

Figura 2. Curva aplicando a base de dados *junit*.

Para isso, foram desenvolvidos experimentos em arquivos de código com uma baixa prevalência de vulnerabilidades. A experimentação permitiu identificar que a reutilização de uma base de dados completa (contendo todos os arquivos rotulados) pode prejudicar o aprendizado do modelo de previsão de vulnerabilidades. Já, ao se adicionar 5 ou 10 arquivos contendo *bugs*, foi observado ganhos promissores em relação à execução original.

Nos trabalhos futuros, pretende-se comparar o método ativo KNEE com outros métodos utilizados para revisões da literatura ([Cormack and Grossman 2016b, Li and Kanoulas 2020]). Além disso, pretende-se analisar experimentalmente outras bases de dados com maior volume de arquivos de código fonte para se entender com mais detalhes os comportamentos divergentes identificados na experimentação.

Referências

[Cormack and Grossman 2016a] Cormack, G. V. and Grossman, M. R. (2016a). Engineering quality and reliability in technology-assisted review. In *ACM SIGIR*, pages 75–84.

[Cormack and Grossman 2016b] Cormack, G. V. and Grossman, M. R. (2016b). Scalability of continuous active learning for reliable high-recall text classification. pages 1039–1048.

[Cruz 2019] 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.

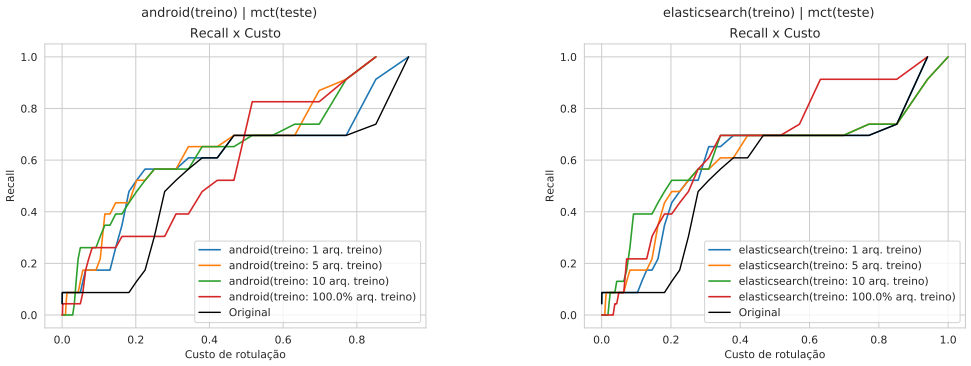

Figura 3. **Curva para a base de dados *mct*.**

[Fisichella et al. 2009] Fisichella, M., Kawase, R., and Gadiraju, U. (2009). Automatic classification of documents in cold-start scenarios.

[Goodfellow and Courville 2016] Goodfellow, Y. and Courville, A. (2016). *Machine Learning Basics*, page 95–160. The MIT Press.

[Li and Kanoulas 2020] 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 Trans. Inf. Syst.*, 38(4).

[Manning et al. 2018] Manning, C. D., Raghavan, P., and Schütze, H. (2018). *Introduction to information retrieval*. Cambridge University Press.

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

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

[Settles 2009b] Settles, B. (2009b). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

[Shamal et al. 2017] 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.

[Tóth et al. 2016] Tóth, Z., Gyimesi, P., and Ferenc, R. (2016). A public bug database of github projects and its application in bug prediction. In *International Conference on Computational Science and Its Applications*, pages 625–638. Springer.

[Yu et al. 2018] 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 et al. 2019] Yu, Z., Theisen, C., Williams, L., and Menzies, T. (2019). Improving vulnerability inspection efficiency using active learning. *IEEE Transactions on Software Engineering*.

[Zhang et al. 2020] Zhang, J., Wu, J., Chen, C., Zheng, Z., and Lyu, M. R. (2020). Cds: A cross–version software defect prediction model with data selection. *IEEE Access*, 8:110059–110072.

[Zhang et al. 2015] Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., and Li, S. (2015). Combining software metrics and text features for vulnerable file prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 40–49. IEEE.