Detecção de Defeitos de Software Através da Aplicação de uma Arquitetura de Rede Neural Profunda

Claudiane Duarte Magalhães[1], **Eder Silva dos Santos Júnior**[1], **D. B. Luque**[1]

[1]Centro de Ciências Exatas e Tecnológicas
Universidade Federal do Acre (UFAC) – Rio Branco, AC – Brasil

{claudiane.magalhaes,eder.junior}@sou.ufac.br, diodomiro.carcasi@ufac.br

Abstract. *The software development lifecycle encompasses the software testing, a rigorous and costly activity. In order to obtain systems quality and reliable software, the adoption of neural networks algorithms rais is a good alternative. Therefore, this article proposes a structure of classification, applied to 12 NASA databases, to detect defects in software. A hybrid resampling technique was used, combining under- sampling and oversampling, aiming to deal with the unbalance problem of classes. The metrics F1-Score, Accuracy, Recall, Precision, MCC and AUC. The results reflected that the proposal outperformed other techniques reported in related works, in the prediction of software defects.*

Resumo. *O ciclo de vida de desenvolvimento de software abrange a etapa de teste de software, uma atividade rigorosa e custosa. Para obter sistemas de software confiáveis e de qualidade, adotar algoritmos de Redes Neurais para predição de possíveis defeitos tem se mostrado uma boa opção. Dessa forma, este artigo propõe uma estrutura de classificação, aplicada em 12 bases de dados da NASA para detectar defeitos de software. Avaliou-se ainda o efeito de duas técnicas de reamostragem híbridas, para tratar o problema de desbalanceamento das classes. Foram utilizadas as métricas acurácia, precisão, sensibilidade, pontuação F1, AUC e MCC, em que foram obtidos resultados competitivos ou superiores a outros estudos.*

1. Introdução

Ao longo dos últimos vinte anos, a demanda por *software* aumentou significamente, para diversas aplicações [Manjula and Florence 2019a]. A fim de atender a essa nova realidade, uma grande quantidade de aplicativos de *software* são desenvolvidos para finalidades do meio coorporativo ou uso diário. Em virtude da produção em massa, a qualidade do *software* continua sendo uma problemática não resolvida que apresenta desempenho insatisfatório, tanto para aplicações industriais quanto individuais [Manjula and Florence 2019a]. Para solucionar esse problema, é introduzida a etapa de teste de *software*, que auxilia a encontrar os defeitos ou bugs em sistemas de *software*, visando posteriormente resolvê-los [Manjula and Florence 2019a].

De acordo com [Neto and Claudio 2007], o teste de *software* é o processo de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no âmbito ao qual foi projetado, com o objetivo de revelar falhas, para que seja possível idenficar e corrigir tais defeitos. Diante disso, Predição de Defeitos de *Software* (do

inglês *Software Defect Prediction* - SDP) é um tópico de pesquisa de Engenharia de Software que visa auxiliar a alocação de dados/recursos de *software* e predizer, com antecedência, módulos de *softwares* propensos a defeitos [Avelino Júnior 2022]. Desse modo, a etapa de teste faz parte do ciclo de vida de desenvolvimento de *software*, que é um ciclo criado para o desenvolvimento de sistemas de *software* confiáveis e de alta qualidade [Afzal and Torkar 2016].

Embora a maioria das etapas do ciclo de vida do *software* sejam conduzidas por profissionais, erros humanos são inevitáveis. Atualmente, esses erros tendem a ser mais intensos, em virtude dos sistemas de *software* modernos serem grandes, com componentes e módulos em uma relação de dependência [Bajeh et al. 2020]. Logo, tais erros, se não corrigidos de imediato, ocasionarão em sistemas de software defeituosos [Balogun et al. 2020].

É importante salientar que a etapa de teste de *software* é uma das atividades mais custosas do processo de desenvolvimento de software, em virtude de ser necessário abranger uma quantidade significativa dos recursos de um projeto [Rana et al. 2014]. O rigor e o custo desta atividade vão depender em especial da complexidade da aplicação a ser desenvolvida, pois distintas aplicações requerem uma preocupação diferenciada com as atividades de teste [Neto and Claudio 2007].

Dessa forma, é necessário levar em consideração a previsão antecipada e detecção de defeitos de *software* antes do lançamento do *software*. A previsão de defeitos de *software* é a adoção de técnicas de Aprendizado de Máquina (do inglês *Machine Learning* - ML) para determinar a defeituosidade de módulos ou componentes de *software* [Catal and Diri 2009], [Mabayoje et al. 2018]. A detecção precoce de módulos ou componentes defeituosos em um sistema de *software* pode garantir a retificação espontânea de tais módulos ou componentes e uso criterioso dos recursos disponíveis [Chauhan and Kumar 2020].

O presente trabalho teve como objetivo treinar e validar uma arquitetura de Rede Neural Profunda em 12 bases de dados de projetos da *National Aeronautics and Space Administration* (NASA), elaboradas por [Shepperd et al. 2013], para detecção de defeitos de *software*. No pré-processamento, foram utilizadas e comparadas duas técnicas de reamostragem híbridas, que combinam *undersampling* e *oversampling*, para tratar o problema de desbalanceamento das classes. Ao final dos testes, analisou-se as métricas de desempenho do modelo, tendo sido selecionadas: Acurácia, Precisão, Sensibilidade, Pontuação F1, MCC e Área sob a Curva ROC (AUC). Essas medidas apresentaram bons resultados, sendo inclusive superiores aos de [Iqbal and Aftab 2020], outro trabalho de predição de defeitos de *software*, por meio de aprendizado profundo, que fez uso dos mesmos conjuntos de dados.

O trabalho está estruturado da seguinte forma: A Seção 2 trata dos trabalhos relacionados; a Seção 3 aborda os materiais e métodos; na Seção 4 são apresentados os resultados e discussões; e, por fim, a Seção 5 contém as conclusões acerca do trabalho.

2. Trabalhos Relacionados

De acordo com [Omri and Sinz 2020] nos últimos anos, foram propostas grandes variedades de modelos de aprendizado profundo, tendo aplicações diversas pela comunidade científica. Consoante com [Herbold et al. 2018], a previsão de defeitos tem se

tornado uma linha de pesquisa frequentemente apresentada para assegurar a garantia de qualidade do *software*, tendo na literatura como primeiros trabalhos empreendidos por [Basili et al. 1996] e [Lanubile et al. 1995], usando técnicas de classificação para prever defeitos de *software*.

Os pesquisadores em [Iqbal et al. 2019b] apresentaram uma estrutura usando técnicas de seleção de atributos e aprendizado de máquina. O *framework* proposto utilizou dois métodos: com seleção de atributos e sem seleção de atributos. O algoritmo foi implementado em 12 conjuntos de dados da NASA. O desempenho é avaliado usando as medidas: Precisão, *Recall*, *F-measure*, Acurácia, MCC e AUC. Os resultados são comparados com as seguintes técnicas: Naive Bayes, Multi-Layer Perceptron, Radial Basis Function, SVM, K Vizinho mais próximo, kStar, OneR, PART, árvore de decisão e floresta aleatória.

Em [Manjula and Florence 2019b], um algoritmo Genético Híbrido foi proposto pelos pesquisadores, baseado em Rede Neural Artificial (RNA) para previsão efetiva de defeitos de *software*. O objetivo do algoritmo é selecionar os recursos ideais e a RNA visa classificar os módulos como defeituosos e não defeituosos. Conjuntos de dados do repositório PROMISE são usados para experimentos e os resultados refletem o melhor desempenho da técnica proposta em comparação com outras técnicas.

O trabalho de [Iqbal and Aftab 2020] apresentou uma estrutura de classificação que usa a técnica de seleção de atributos *MultiFilter* e uma arquitetura *Multi-Layer Perceptron* (MLP) para prever módulos de softwares propensos a defeitos. A estrutura consistiu em duas versões: 1) com oversampling, 2) sem oversampling. A sobreamostragem é introduzida na estrutura para analisar o efeito do desequilíbrio de classe. A estrutura é implementada usando 12 conjuntos de dados da NASA e o desempenho é avaliado através das métricas: F-measure, Acurácia, MCC e ROC.

3. Materiais e Métodos

Esta seção aborda os materiais e métodos utilizados. Foi empregada a metodologia geral descrita na Figura 1. A primeira etapa da metodologia proposta foi a seleção de um *Dataset*, ou base de dados, relevante. Após selecionado, o conjunto de dados foi submetido a uma etapa de pré-processamento, passando por uma normalização, balanceamento das classes, e pela divisão da base reamostrada em treino e teste. Em seguida, o algoritmo de classificação, que é a rede neural utilizada, pôde ser treinado e validado, sendo extraídos os resultados do desempenho do modelo.

A estrutura proposta foi implementada no *software* Anaconda Navigator, que se trata de um Ambiente de Desenvolvimento Integrado (do inglês *Integrated Development Environment* - IDE). No IDE, optou-se pela utilização da ferramenta *jupyterlab*, onde é possível programar em linguagem *Python*, escolhida para o projeto, por ser amplamente utilizada no campo de Ciência de Dados e *Machine Learning*, graças ao advento de bibliotecas como *Pandas*, *TensorFlow* e *Keras* [Homem and Ufes 2020].

3.1. Bases de Dados

A aquisição e organização dos dados é uma das partes mais importantes no Aprendizado de Máquina, pois, considerando os tipos de dados, define-se aspectos importantes da arquitetura do modelo [Beduin 2021]. A estrutura proposta no trabalho foi implementada

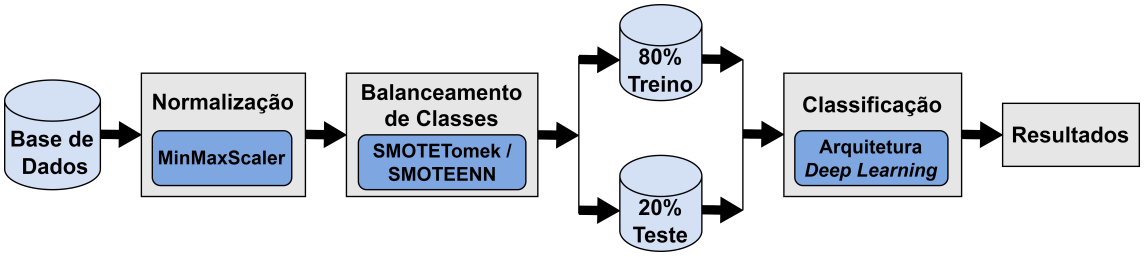

Figura 1. Etapas do método proposto.

em 12 conjuntos de dados da NASA, elaborados por [Shepperd et al. 2013], para testes de detecção de defeitos de *software*. A escolha destas bases de dados foi motivada pelo seu uso em diversos estudos similares, estando presentes em [Iqbal and Aftab 2020], [Iqbal et al. 2019a] e [Manjula and Florence 2019b], por exemplo.

Os conjuntos de dados são: CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4 e PC5, que são descritos detalhadamente na Tabela 1. Esta Tabela contém a quantidade de atributos de cada base, o número de instâncias, de *softwares* defeituosos e não-defeituosos, além do percentual de módulos com defeito, as linguagens de programação utilizadas nos projetos e o número total de linhas de cada código fonte.

Cada base de dados utilizada representa um sistema de *software* específico da NASA e consiste em vários atributos/recursos junto com a classe de saída conhecida (classe de destino). A classe de destino/saída é o atributo dependente e os atributos restantes são conhecidos como atributos independentes. No caso, a classe de destino nos conjuntos de dados tem um dos seguintes valores: “Y” ou “N”, onde “Y” significa que a instância especificada (ou módulo) é defeituosa e “N” significa que não é defeituosa.

Os pesquisadores em [Shepperd et al. 2013] criaram duas versões dessas bases de dados da NASA: D’ (inclui instâncias duplicadas e inconsistentes) e D” (Não inclui instâncias duplicadas e inconsistentes). Em particular, neste trabalho, foi selecionada a versão D”, por estar em condições mais apropriadas para os testes.

Tabela 1. Descrição detalhada das bases de dados.

Base de Dados	Atributos	Instâncias	Defeito	Não-Defeito	Defeito (%)	Linguagem	N° de Linhas
CM1	38	327	42	285	12.8	C	20k
JM1	22	7.720	1,612	6,108	20.8	C	315k
KC1	22	1.162	294	868	25.3	C++	43k
KC3	40	194	36	158	18.5	Java	18k
MC1	39	1.952	36	1916	1.8	C++	63k
MC2	40	124	44	80	35.4	C	6k
MW1	38	250	25	225	10	C	8k
PC1	38	679	55	624	8.1	C	40k
PC2	37	722	16	706	2.2	C	26k
PC3	38	1.053	130	923	12.3	C	40k
PC4	38	127	176	1094	13.8	C	36k
PC5	9	1.694	458	1236	27.0	C++	16k

3.2. Pré-Processamento dos Dados

Nesta etapa, foi aplicado um tipo de normalização para cada base de dados, além das técnicas de reamostragem selecionadas para contornar o desbalanceamento das classes. Em particular, utilizou-se a normalização *MinMaxScaler*, disponível na biblioteca *sklearn* do *Python*. Esta normalização redimensiona os valores dos atributos de tal forma que fiquem limitados ao intervalo (0,1) [Kramer 2016].

A respeito das técnicas de reamostragem, optou-se por métodos híbridos que combinam *undersampling* e *oversampling*, por serem estratégias mais avançadas no combate ao desbalanceamento de classes. No caso, foram selecionados o SMOTEEN e o SMOTETomek, que juntam o SMOTE com duas técnicas de *undersampling*, que são o TomekLinks e o nearest-neighbours [Vaz 2019].

3.3. Arquitetura de Rede Neural Profunda Utilizada

Segundo [Nascimento 2003], as Redes Neurais fundamentam-se em estudos sobre a estrutura do cérebro humano, constituindo um sistema de circuitos que buscam simular sua forma inteligente de processar informação, inclusive seu comportamento, aprendendo, errando e fazendo novas descobertas.

Conforme [Haykin 2000] foi inspirado nessas características biológicas que surgiu a ideia de redes neurais artificiais, que podem ser implementadas através de componentes eletrônicos ou por programação em um computador digital. Essas redes neurais são formadas por unidades elementares de processamento denominadas ”neurônios artificiais”, cuja nomenclatura é uma alusão aos neurônios biológicos, em que foram baseadas [Nascimento 2003].

O primeiro modelo de neurônio artificial foi proposto por [Rosenblatt 1960], uma máquina de aprendizado que ficou conhecida como *perceptron* e que, inicialmente, era constituída de uma única camada. Porém, posteriormente foi criado em [Rumelhart 1986] um novo método de treinamento direcionado a *perceptrons* multicamadas, chamado de aprendizado por retropropagação (do inglês *backpropagation*), que trouxe consigo um grande impacto científico. Muitos outros trabalhos deram continuidade a ideia, fazendo com que as redes neurais representem hoje uma área de pesquisa multidisciplinar [Nascimento 2003].

Este trabalho utiliza uma arquitura de rede neural multicamadas, ilustrada na Figura 2. Conforme [Kovács 2002], uma rede neural básica possui três camadas interconectadas, que são a camada de entrada, camada oculta e camada de saída. A camada de entrada é constituída pelos neurônios que recebem diretamente as entradas da rede, de modo que os dados são processados ao longo da rede até atingirem a camada final, que é a camada de saída. Já as camadas internas, que não são nem a de entrada e nem a de saída, recebem o nome de camadadas ocultas. Em particular, a rede neural utilizada possui três camadas ocultas, cada uma com 20 neurônios (n_1, n_2, ..., n_{20}), e uma camada de saída, com função de ativação *sigmóide*. Esta função de ativação, dada pela Equação 1, retorna um valor no intervalo $[0, 1]$, sendo muito útil para prever uma probabilidade como saída [Gaio 2022].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Em relação aos hiperparâmetros de treinamento do modelo, adotou-se como otimizador o *Stochastic Gradient Descent* (SGD), com a acurácia como métrica de otimização, *binary cross-entropy* como função de custo, um *batch size* variando entre 1, 4 e 16, e um total de 550 épocas de treinamento, como mostra a Tabela 2.

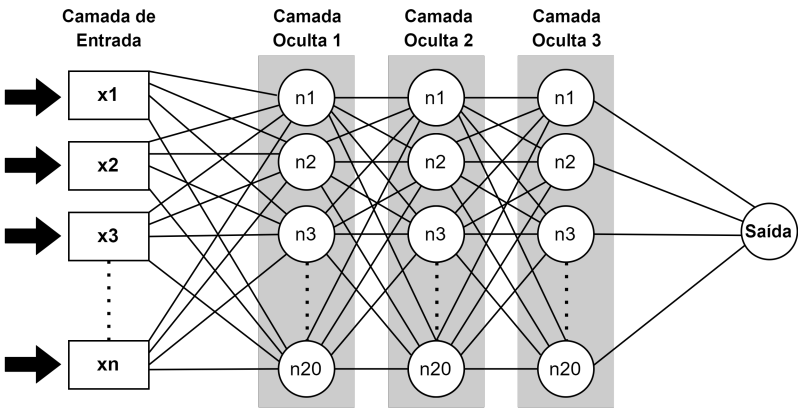

Figura 2. Ilustração da rede neural utilizada.

Tabela 2. Configuração de treinamento.

Hiperparâmetros de Treinamento	
Otimizador	*sgd*
Métrica de Otimização	*accuracy*
Função de Custo	*binary_crossentropy*
Batch Size	*1, 4, 16*
Épocas	*550*

3.4. Métricas de Validação

Segundo [Monard and Baranauskas 2003] a matriz de confusão possui um papel essencial na visualização da assertividade de modelos em tarefas de classificação, já que ela correlaciona as classes reais com as preditas, conforme ilustrado na Figura 3. Sendo TP a quantidade de Verdadeiros Positivos (do inglês *True Positives*), TN a quantidade de Verdadeiros Negativos (do inglês *True Negatives*), FP a quantidade de Falsos Positivos (do inglês *False Positives*) e FN a quantidade de Falsos Negativos (do inglês *False Negatives*).

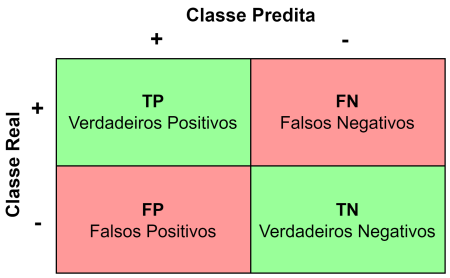

Figura 3. Ilustração de uma matriz de confusão

As métricas de desempenho da etapa de validação podem ser determinadas a partir da matriz de confusão. Neste estudo, foram utilizadas a acurácia (*accuracy* ou ACC),

precisão (*Precision*), sensibilidade (*Recall*), pontuação F1 (*F1-Score*), Área sob a Curva ROC (AUC) e *Matthews Correlation Coefficient* (MCC). As métricas, definidas de acordo com [Iqbal and Aftab 2020], são explicadas a seguir:

- ACC: A Acurácia, em problemas de classificação, é o número de previsões corretas feitas pelo modelo sobre todos os tipos de previsões feitas. Essa métrica fornece um medida de desempenho geral do modelo e pode ser calculada por meio da Equação 2.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

- *Precision*: A precisão é a proporção de verdadeiros positivos (TP) em relação ao número total de amostras que classificadas como positivo. Para calcular a precisão a respeito de uma determinada classe (por exemplo, classe positiva), é utilizada a Equação 3.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

- *Recall*: É a proporção entre as previsões corretas e o total de amostras daquela classe sendo avaliadas. Essa métrica indica quantos exemplos de um classe, em relação ao total presente no conjunto, foram identificados pelo modelo. A sensibilidade é dada pela Equação 4.

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

- *F1-Score*: É a média harmônica entre a precisão e a sensibilidade. Uma vez que seu valor está alto, significa que a acurácia obtida é relevante, ou seja, os valores de TP, TN, FP e FN aferidos não apresentam grandes distorções. Desse modo, essa métrica também pode ser interpretada como uma medida de confiabilidade da acurácia.

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5}$$

- AUC: É uma métrica que diz a área sob a curva da Característica de Operação do Receptor (do inglês *Receiver Operating Characteristic* - ROC), a partir da taxa de verdadeiros positivos (TP_r) e da taxa de falsos positivos (FP_r).

$$AUC = \frac{2 + TP_r - FP_r}{2} \tag{6}$$

- MCC: É uma métrica que reflete a proporção entre as classificações observadas e as classificações previstas. O MCC é uma métrica robusta para experimentos envolvendo bancos de dados desbalanceados.

$$MCC = \frac{TN \cdot TP - FN \cdot FP}{\sqrt{(FP + TP) \cdot (FN + TP) \cdot (TN + FP) \cdot (TN + FN)}} \tag{7}$$

Tabela 3. Comparação dos resultados para cada técnica de reamostragem.

Bases	Reamostragem	Recall	ACC	Precision	F1-Score	MCC	AUC
CM1	**SMOTEENN**	**1,00**	**0,98**	**0,97**	**0,99**	**0,97**	**0,98**
	SMOTETomek	0,93	0,91	0,92	0,92	0,82	0,91
JM1	**SMOTEENN**	**0,90**	**0,72**	**0,72**	**0,80**	**0,38**	**0,66**
	SMOTETomek	0,63	0,65	0,66	0,65	0,30	0,65
KC1	**SMOTEENN**	**0,91**	**0,88**	**0,89**	**0,90**	**0,74**	**0,87**
	SMOTETomek	0,67	0,73	0,77	0,71	0,43	0,73
KC3	**SMOTEENN**	**0,98**	**0,96**	**0,95**	**0,97**	**0,91**	**0,95**
	SMOTETomek	0,93	0,86	0,80	0,86	0,73	0,86
MC1	**SMOTEENN**	**1,00**	**1,00**	**0,99**	**1,00**	**0,99**	**0,99**
	SMOTETomek	1,00	0,98	0,97	0,98	0,97	0,98
MC2	**SMOTEENN**	**0,94**	**0,93**	**0,94**	**0,94**	**0,85**	**0,92**
	SMOTETomek	0,95	0,87	0,78	0,86	0,75	0,88
MW1	**SMOTEENN**	**1,00**	**0,98**	**0,97**	**0,98**	**0,96**	**0,98**
	SMOTETomek	0,98	0,93	0,89	0,94	0,87	0,93
PC1	**SMOTEENN**	**0,99**	**0,97**	**0,96**	**0,98**	**0,94**	**0,97**
	SMOTETomek	**0,99**	**0,97**	**0,96**	**0,97**	**0,95**	**0,97**
PC2	**SMOTEENN**	**1,00**	**0,99**	**0,99**	**0,99**	**0,98**	**0,99**
	SMOTETomek	**1,00**	0,98	0,98	0,98	0,96	0,98
PC3	**SMOTEENN**	**1,00**	**0,97**	**0,95**	**0,97**	**0,93**	**0,96**
	SMOTETomek	0,96	0,92	0,89	0,93	0,85	0,92
PC4	**SMOTEENN**	**1,00**	**0,97**	**0,96**	**0,98**	**0,95**	**0,97**
	SMOTETomek	0,97	0,94	0,91	0,94	0,88	0,94
PC5	**SMOTEENN**	**1,00**	**0,93**	**0,89**	**0,94**	**0,86**	**0,92**
	SMOTETomek	0,85	0,79	0,75	0,80	0,58	0,79

4. Resultados e Discussões

Na Tabela 3 são apresentados os resultados das métricas de validação, para os diferentes *datasets*, considerando as duas técnicas de reamostragem que foram utilizadas. Nota-se que o método SMOTEENN demonstrou ser superior ao SMOTETomek, indicando que esse tipo avançado de balanceamento faz com que o modelo possa aprender melhor os padrões da classe 1, melhorando sua capacidade de detectar defeitos de *software*.

Na Tabela 4, tem-se uma comparação entre os resultados alcançados com a técnica SMOTEEN, que proporcionou melhores resultados, e o trabalho de [Iqbal and Aftab 2020], adotado como referência. Os valores das métricas deixam claro que uma arquitetura mais profunda, como a que foi utilizada no presente trabalho, pode produzir modelos com desempenhos superiores. Somando-se a isso, é visível que a técnica híbrida de reamostragem faz com que o algoritmo aprenda mais a respeito da classe minoritária, sendo este método mais avançado que o *Random Over Sampling* (ROS), que foi aplicado em [Iqbal and Aftab 2020].

Vale ressaltar que em [Iqbal and Aftab 2020] os autores realizaram uma operação de seleção de atributos (do inglês *Feature Selection* - FS), que o este trabalho optou por não realizar. Esta decisão teve o objetivo de avaliar a viabilidade de uma metodologia

mais simples e com desempenho igual ou superior.

Tabela 4. Comparação deste trabalho com [Iqbal and Aftab 2020].

Bases	ACC		F1-Score		MCC		AUC	
	Este Trabalho	Iqbal, Aftab (2020)	**Este Trabalho**	Iqbal, Aftab (2020)	**Este Trabalho**	Iqbal, Aftab (2020)	**Este Trabalho**	Iqbal, Aftab (2020)
CM1	**0,98**	0,79	**0,99**	0,80	**0,97**	0,82	**0,98**	0,81
JM1	**0,72**	0,62	**0,80**	0,55	**0,38**	0,27	0,66	**0,68**
KC1	**0,88**	0,62	**0,90**	0,64	**0,74**	0,25	**0,87**	0,70
KC3	**0,96**	0,63	**0,97**	0,58	**0,91**	0,35	**0,95**	0,73
MC1	**1,00**	0,83	**1,00**	0,85	**0,99**	0,68	**0,99**	0,90
MC2	**0,93**	0,75	**0,94**	0,66	**0,85**	0,53	**0,92**	0,68
MW1	**0,98**	0,77	**0,98**	0,79	**0,96**	0,54	**0,98**	0,68
PC1	**0,97**	0,89	**0,98**	0,90	**0,94**	0,79	**0,97**	0,95
PC2	**0,99**	0,91	**0,99**	0,91	**0,98**	0,83	**0,99**	0,92
PC3	**0,97**	0,75	**0,97**	0,78	**0,93**	0,54	**0,96**	0,83
PC4	**0,97**	0,84	**0,98**	0,84	**0,95**	0,70	**0,97**	0,92
PC5	**0,93**	0,70	**0,94**	0,73	**0,86**	0,42	**0,92**	0,77

Observa-se que os modelos se detacaram em todas as bases de dados utilizadas, como indicam as marcações em negrito na Tabela 4. Pela análise individual das métricas de desempenho, nota-se que a acurácia máxima foi de 100%, atingida para a base MC1, e que todas as demais foram expressivas, sendo a maioria acima de 90%. Do ponto de vista de *F1-Score*, que está relacionada à precisão e sensibilidade, também é visível o quanto os resultados foram altos. As médias harmônicas obtidas demonstram que tanto a precisão quanto a sensibilidade foram altas, como desejado. Sabe-se que na predição de defeitos de *software* é essencial ter uma boa sensibilidade, para que a quantidade de Falsos Negativos seja mínima.

Por fim, a métrica MCC enfatiza que o problema de desbalanceamento das classes foi bem contornado. Em muitos casos, foram encontrados valores de AUC e MCC muito próximos a 100%. Essas medidas dão maior confiabilidade à acurácia dos modelos, pois reforçam o quanto ele é robusto e generalizável. Especificamente o MCC é uma métrica que é robusta diante de bases de dados desbalanceadas, além de ser uma medida que envolve todas as medidas de classificação possíveis, ou seja, TP, TN, FP e FN.

5. Conclusões

Este trabalho apresentou uma estrutura de classificação baseada em uma arquitetura de Rede Neural Profunda, para previsão de defeitos de *software*. O método proposto consistiu na seleção de um conjunto de dados, em seguida, foi feito o pré-processamento dos dados, incluindo a separação em treino e teste, e o modelo foi treinado e validado, sendo obtidos os resultados referentes ao seu desempenho.

Em relação à primeira etapa, de seleção dos dados, optou-se pela lista de *datasets* de projetos da NASA criada por[Shepperd et al. 2013], pois as bases de dados passaram por um tratamento e se encontram livres de atributos com valores faltantes, duplicados e

inconsistentes. Contudo, esses conjuntos de dados não deixam de estar desbalanceados, tal que foi necessário, para melhoria das métricas do modelo, o uso de técnicas de balanceamento de classes. Neste estudo, duas técnicas categorizadas como híbridas foram utilizadas (SMOTEEN e SMOTETomek), para experimentação de um método de maior complexidade, que não foi empregado por [Iqbal and Aftab 2020], que utilizou o mesmo conjunto de *datasets*.

Dito isto, o trabalho teve como um de seus objetivos comparar o efeito de duas abordagens de reamostragem, onde foi possível concluir que o SMOTEENN demonstra ser uma estratégia superior ao SMOTETomek, com base nas métricas de desempenho do modelo, em que foram analisados a Acurácia, Precisão, Sensibilidade, Pontuação F1, MCC e AUC. Ademais, verifica-se que os modelos, com a fusão do balanceamento de dados com aprendizado profundo, apresentou métricas de desempenho inclusive maiores àquelas obtidas por [Iqbal and Aftab 2020], outro trabalho de predição de defeitos de *software* que fez uso dos mesmos conjuntos de dados. Nota-se que a maioria dos modelos apresentou uma acurácia maior que 90% e altos valores de *F1-Score*, AUC e MCC, superiores aos de [Iqbal and Aftab 2020].

Em suma, as principais contribuições deste trabalho são: (a) Utilização de uma nova estrutura de classificação de defeitos de *software*, (b) Comparação entre duas técnicas híbridas de reamostragem, aplicadas a uma base de dados desbalanceada, (c) treinamento e validação de uma arquitetura de *deep learning* com três camadas ocultas de 20 neurônios em 12 bases de dados de predição de defeitos de *software*, (d) obtenção de resultados competitivos ou superiores a outros estudos envolvendo *deep learning*, agregando à literatura de ES modelos promissores para implementação e novos testes de predição de defeitos de *softwares*.

Para trabalhos futuros, é interessante investigar como outros modelos de aprendizado profundo podem afetar os resultados, aplicando a mesma estrutura proposta neste trabalho em outras bases de dados, com mais instâncias e atributos. Sugere-se também a investigação da previsão do número de defeitos em módulos de *software* incluindo mais projetos escritos em diferentes linguagens de programação e projetos comerciais da indústria. Ainda como trabalhos futuros, poderiam ser desenvolvidos novos conjuntos de dados com o auxílio de engenheiros de teste de *software*, pois no campo de previsão de defeitos, são necessários conjuntos de dados novos e comerciais adicionais. Também seria interessante investigar novos critérios de desempenho para modelos de previsão de defeitos de *software*.

Agradecimentos

Os autores agradecem ao suporte financeiro da CAPES por meio do Programa de Pós-graduação em Ciência da Computação (PPgCC) da UFAC.

Referências

Afzal, W. and Torkar, R. (2016). Towards benchmarking feature subset selection methods for software fault prediction. *Computational intelligence and quantitative software engineering*, pages 33–58.

Avelino Júnior, J. S. (2022). Uma abordagem de seleção dinâmica de classificadores para predição de defeitos de software. Master's thesis, Universidade Federal de Pernambuco.

Bajeh, A. O., Oluwatosin, O.-J., Basri, S., Akintola, A. G., and Balogun, A. O. (2020). Object-oriented measures as testability indicators: An empirical study. *J. Eng. Sci. Technol*, 15:1092–1108.

Balogun, A., Bajeh, A., Mojeed, H., and Akintola, A. (2020). Software defect prediction: A multi-criteria decision-making approach. *Nigerian Journal of Technological Research*, 15(1):35–42.

Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751–761.

Beduin, I. R. O. (2021). Detecção da covid-19 em imagens de raio-x: construindo um novo modelo de aprendizado profundo utilizando automl.

Catal, C. and Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040–1058.

Chauhan, A. and Kumar, R. (2020). Bug severity classification using semantic feature with convolution neural network. In *Computing in Engineering and Technology: Proceedings of ICCET 2019*, pages 327–335. Springer.

Gaio, D. E. (2022). Análise comparativa das técnicas de implementação de arquiteturas da função sigmoide.

Haykin, S. S. (2000). Redes neurais artificiais: princípio e prática. *2ª Edição, Bookman, São Paulo, Brasil*.

Herbold, S., Trautsch, A., and Grabowski, J. (2018). A comparative study to benchmark cross-project defect prediction approaches. In *Proceedings of the 40th International Conference on Software Engineering*, pages 1063–1063.

Homem, W. L. and Ufes, P. E. M. (2020). Apostila de machine learning. *PET Engenharia Mecânica, UFES*.

Iqbal, A. and Aftab, S. (2020). A classification framework for software defect prediction using multi-filter feature selection technique and mlp. *International Journal of Modern Education & Computer Science*, 12(1).

Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., and Husen, A. (2019a). Performance analysis of machine learning techniques on software defect prediction using nasa datasets. *International Journal of Advanced Computer Science and Applications*, 10(5).

Iqbal, A., Aftab, S., Ullah, I., Bashir, M. S., and Saeed, M. A. (2019b). A feature selection based ensemble classification framework for software defect prediction. *International Journal of Modern Education and Computer Science*, 11(9):54.

Kovács, Z. L. (2002). *Redes neurais artificiais*. Editora Livraria da Fisica.

Kramer, O. (2016). Scikit-learn. *Machine learning for evolution strategies*, pages 45–53.

Lanubile, F., Lonigro, A., and Vissagio, G. (1995). Comparing models for identifying fault-prone software components. In *SEKE*, pages 312–319. Citeseer.

Mabayoje, M. A., Balogun, A. O., Bajeh, A. O., and Musa, B. A. (2018). Software defect prediction: effect of feature selection and ensemble methods.

Manjula, C. and Florence, L. (2019a). Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*, 22(Suppl 4):9847–9863.

Manjula, C. and Florence, L. (2019b). Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*, 22(Suppl 4):9847–9863.

Monard, M. C. and Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, 1(1):32.

Nascimento, J. P. R. (2003). Análise e classificação de imagens baseadas em características de textura utilizando matrizes de co-ocorrência.

Neto, A. and Claudio, D. (2007). Introdução a teste de software. *Engenharia de Software Magazine*, 1:22.

Omri, S. and Sinz, C. (2020). Deep learning for software defect prediction: A survey. In *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*, pages 209–214.

Rana, R., Staron, M., Hansson, J., and Nilsson, M. (2014). Defect prediction over software life cycle in automotive domain state of the art and road map for future. In *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, pages 377–382. IEEE.

Rosenblatt, F. (1960). Perceptron simulation experiments. *Proceedings of the IRE*, 48(3):301–309.

Rumelhart, D. (1986). E., hinton ge, and willians rj. *Learning representations by back-propagating errors", Nature*, 323:6188.

Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215.

Vaz, A. L. (2019). Como lidar com dados desbalanceados em problemas de classificação. Disponível online: `https://medium.com/@arthurlambletvaz`. Acesso em: 19 de fevereiro de 2023.