Research, Society and Development, v. 11, n. 3, e55411326984, 2022
(CC BY 4.0) | ISSN 2525-3409 | DOI: http://dx.doi.org/10.33448/rsd-v11i3.26984

Algoritmo Genético e PSO Aplicados à Escolha dos Hiperparâmetros de uma Rede Neural MLP para Classificação de Requisitos Não-Funcionais

Genetic Algorithm and PSO Applied to the Choice of Hyperparameters of an MLP Neural Network for Non-Functional Requirements Classification

Algoritmo Genético y PSO Aplicados a la Elección de los Hiperparámetros de una Red Neuronal MLP para la Clasificación de Requisitos no Funcionales

Recebido: 14/02/2022 | Revisado: 21/02/2022 | Aceito: 25/02/2022 | Publicado: 07/03/2022

Thiago Matheus Torres Buarque
ORCID: https://orcid.org/0000-0002-5146-6789
Universidade Católica de Pernambuco, Brasil
E-mail: thiago.2019206636@unicap.br
Matheus Barreto Lins Marinho
ORCID: https://orcid.org/0000-0002-4383-625X
Universidade de Pernambuco, Brasil
E-mail: mblm@ecomp.poli.br
Francisco Madeiro Bernardino Junior
ORCID: https://orcid.org/0000-0002-6123-0390
Universidade Católica de Pernambuco, Brasil
E-mail: francisco.madeiro@unicap.br

Resumo
Os requisitos não-funcionais (RNFs) desempenham um papel importante na área de Engenharia de Software (ES), estando associados à construção, ao funcionamento e à manutenção de uma aplicação de qualidade. O sucesso da tarefa manual de classificação de RNFs depende do conhecimento e da experiência do engenheiro de requisitos, além de demandar tempo. Trabalhos têm sido desenvolvidos visando a aplicação de algoritmos de aprendizagem de máquina para classificar automaticamente RNFs, cenário em que devem ser escolhidos os hiperparâmetros do modelo classificador. Neste trabalho, o Algoritmo Genético (GA, *Genetic Algorithm*) e o algoritmo de Otimização por Enxame de Partículas (PSO, *Particle Swarm Optimization*) são utilizados para encontrar hiperparâmetros da rede Neural Perceptron Multicamada (MLP, *Multilayer Perceptron*), com o objetivo de classificar RNFs presentes no conjunto de dados PROMISE_exp. O GA encontrou uma combinação de hiperparâmetros que resultou em F1 de 0,6349, enquanto o PSO encontrou uma combinação que obteve 0,6426 de F1.
Palavras-chave: Classificação; Requisitos de software; Algoritmo genético; PSO; Perceptron multicamadas.

Abstract
Non-functional requirements (NFRs) play an important role in the Software Engineering (SE) area, being associated with the construction, operation, and maintenance of a quality application. The success of the RNF classification manual task depends on the knowledge and experience of the requirements engineer and is time-consuming. Works have been developed aiming at the application of machine learning algorithms to automatically classify RNFs, a scenario in which the hyperparameters of the classifier model must be chosen. In this work, the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO) algorithm are used to find hyperparameters of the Multilayer Neural Perceptron Network (MLP), with the objective of classifying NFRs present in the PROMISE_exp dataset. The GA found a combination of hyperparameters that gave an F1 of 0.6349, while the PSO found a combination that got 0.6426 of F1.
Keywords: Classification; Software requirements; Genetic Algorithm; PSO; Multilayer Perceptron.

Resumen
Los requisitos no funcionales (RNFs) juegan un papel importante en el área de Ingeniería de Software (IS), estando asociados con la construcción, operación y mantenimiento de una aplicación de calidad. El éxito de la tarea del manual de clasificación RNF depende del conocimiento y la experiencia del ingeniero de requisitos y requiere mucho tiempo. Se han desarrollado trabajos encaminados a la aplicación de algoritmos de aprendizaje automático para clasificar automáticamente los RNF, escenario en el que se deben elegir los hiperparámetros del modelo clasificador. En este trabajo, el Algoritmo Genético (GA, *Genetic Algorithm*) y el algoritmo de Optimización de Enjambre de Partículas (PSO, *Particle Swarm Optimization*) se utilizan para encontrar hiperparámetros de la Red de Perceptrón Neural Multicapa (MLP, *Multilayer Perceptron*), con el objetivo de clasificar los RNF presentes en el PROMISE_exp conjunto

de datos. La GA encontró una combinación de hiperparámetros que dio un F1 de 0.6349, mientras que el PSO encontró una combinación que obtuvo 0.6426 de F1.
Palabras clave: Clasificación; Requisitos de software; Algoritmo genético; PSO; Perceptrón multicapa.

1. Introdução

Os Requisitos de Software (RS) representam as necessidades e comportamentos de uma aplicação e, quando bem descritos, são a base para a construção de uma aplicação de qualidade (Navarro-Almanza et al., 2017). Os RS são classificados em Requisitos Funcionais (RFs) e Requisitos Não-Funcionais (RNFs). Os RFs descrevem as funcionalidades enquanto os RNFs descrevem as propriedades e limitações do sistema (Kurtanović & Maalej, 2017).

Os RNFs são divididos em classes, como segurança, desempenho e disponibilidade, e possuem uma posição crítica com relação à qualidade final do sistema (Casamayor et al., 2010). A correta classificação dos RNFs está ligada à facilidade em realizar futuras alterações nos requisitos, evitando, assim, gastos além do previsto e até mesmo a invalidação do projeto, como aconteceu com o sistema de compartilhamento de inteligência do exército dos Estados Unidos, o *Distributed Common Ground System-Army* (DCGS-A). Avaliado em 2,7 bilhões de dólares, o sistema foi desativado devido a problemas de capacidade, desempenho e usabilidade (Hoskinson, 2011).

Uma abordagem comum para classificar automaticamente RS é a aprendizagem supervisionada precedida do pré-processamento dos requisitos (Abad et al., 2017; Baker et al., 2019; Canedo & Mendes, 2020; Iqbal et al., 2018; Kurtanović & Maalej, 2017; Navarro-Almanza et al., 2017; Slankas & Williams, 2013). Nesse cenário, é imprescindível a escolha apropriada dos hiperparâmetros do modelo para obter um resultado satisfatório.

Este trabalho utiliza o Algoritmo Genético (GA, *Genetic Algorithm*) e o algoritmo de Otimização por Enxame de Partículas (PSO, *Particle Swarm Optimization*) para encontrar hiperparâmetros da rede neural Perceptron Multicamadas (MLP, *Multilayer Perceptron*) com o objetivo de classificar RNFs presentes na base de dados PROMISE_exp. Esses dois algoritmos foram escolhidos com base em trabalhos anteriores de otimização de hiperparâmetros (Carvalho & Ludermir, 2007; Kumar et al., 2021; Yoo et al., 2019; Zhao & Qian, 2011).

2. Metodologia

De acordo com o teorema "*No Free Lunch*", apresentado por Wolpert e Macready (1997), não há um algoritmo que satisfaça todos os problemas de otimização, semelhantemente não existe uma única combinação de hiperparâmetros que satisfaça todos os problemas de otimização. Vale ressaltar, também, que não existe uma única combinação de hiperparâmetros satisfatória para o problema abordado. Tendo em vista o mencionado para o cenário de classificação de RNFs usando MLP, o GA e o PSO são utilizados com o objetivo de encontrar combinações de hiperparâmetros da rede MLP entre os valores e intervalos apresentados na Tabela 1.

Tabela 1 - Espaço de busca dos hiperparâmetros.

Hiperparâmetro	Valores/Intervalo de valores
Algoritmo otimizador	{RMSprop, Adam, Adamax}
Função de ativação oculta	{Sigmoid, Tanh, Relu, Elu}
Quantidade de neurônios ocultos	[32 ... 1024]
Taxa de aprendizado	[0,001 ... 0.1]
Dropout	[0,1 ... 0,5]
Tamanho do mini-lote	{32, 64}

Fonte: Elaborada pelos autores.

As implementações foram feitas na linguagem de programação Python (https://python.org) mediante utilização da API Keras (https://keras.io) para implementação da rede MLP e dos algoritmos de otimização. Para a camada de saída da rede MLP, a função de ativação utilizada foi a Softmax (Sharma et al., 2017).

2.1 Pré-processamento

Para o pré-processamento, os requisitos foram convertidos para minúsculo e removidos os símbolos, caracteres especiais e números. As técnicas de remoção de palavras irrelevantes (*Stopwords*) e Lematização foram aplicadas utilizando a biblioteca NLTK (https://www.nltk.org/). O Modelo de Espaço Vetorial (MEV) escolhido foi o *Term Frequency — Inverse Document Frequency* (TF-IDF). A escolha desse MEV teve como base o desempenho obtido na classificação de requisitos de *software* no trabalho realizado por Canedo e Mendes (2020). O TF-IDF foi implementado utilizando a API Keras.

2.2 Conjunto de dados

O conjunto de dados PROMISE_exp (https://tinyurl.com/PROMISE-exp) é uma expansão do tera_PROMISE (https://terapromise.csc.ncsu.edu/!/#repo/view/head/requirements/nfr) realizada por Lima et al. (2019). Esse conjunto de dados é caracterizado por 969 requisitos de software, sendo 444 funcionais e 525 não-funcionais, escritos em língua inglesa, divididos em 11 classes. A Tabela 2 apresenta a divisão de classes do PROMISE_exp e a respectiva quantidade de amostras de cada classe.

Tabela 2 - Divisão dos RS presentes no conjunto de dados PROMISE_exp.

Classe	Quantidade
Funcional	444
Disponibilidade	31
Tolerância a falhas	18
Legal	15
Aparências e comportamento	49
Manutenibilidade	24
Operacional	77
Performance	67
Portabilidade	12
Escalabilidade	22
Segurança	125
Usabilidade	85
Total	969

Fonte: Elaborada pelos autores.

2.3 Avaliação do classificador

Cada combinação de hiperparâmetros gerada durante as execuções do GA e do PSO caracteriza um classificador MLP, avaliado mediante validação cruzada com uso de 10 dobras (*k-fold*=10) e treinamento de no máximo 50 épocas. A cada dobra (iteração), 10% do total de RNFs foram removidos do conjunto de treino para compor o conjunto de validação, monitorado pelo método de parada antecipada com tolerância de cinco épocas. O valor de aptidão do modelo foi dado pela média das medidas F1 obtidas nas dez dobras *k-fold*.

2.4 Algoritmo Genético

O Algoritmo Genético foi implementado utilizando a biblioteca DEAP (https://deap.readthedocs.io/en/master/), utilizando a abordagem do Elitismo (Baluja & Caruana, 1995) para garantir que os 3 melhores cromossomos (10% da população) estivessem na próxima geração sem sofrer qualquer alteração. A seleção por torneio (Bäck et al., 1999) foi utilizada para obter os demais indivíduos a serem submetidos à próxima etapa do GA. O cruzamento utilizado foi o de dois pontos e, para a fase de

mutação, apenas um gene aleatório foi escolhido. A população resultante do processo de mutação é a nova população. O GA foi executado 30 vezes com os valores dos parâmetros apresentados na Tabela 3.

Tabela 3 - Parâmetros do GA.

Parâmetro	Valor
Probabilidade de cruzamento	95%
Probabilidade de mutação	10%
Tamanho da população	30
Gerações	20

Fonte: Elaborada pelos autores.

2.5 Otimização por Enxame de Partículas

O PSO foi implementado utilizando o decaimento linear proposto por Xin et al. (2009) para definir o valor do coeficiente de inércia w. Os autores supracitados realizaram diversos experimentos e comprovaram que um decaimento linear entre 0,9 e 0,4, no geral, acarreta resultados satisfatórios. O decaimento linear (Equação 1) é utilizado na atualização de velocidade da partícula (Equação 2)

$$w = (w_{max} - w_{min}) * (t_{max} - t)/t_{max} + w_{min}, \quad (1)$$

$$v_{ij}(t+1) = w * v_{ij}(t) + C_1 * r1 * (y_{ij}(t) - x_{ij}(t) + C_2 * r2 * (\hat{y}_j(t) - x_{ij}(t))) \quad (2)$$

em que w_{max} é o valor máximo do coeficiente de inércia, w_{min} o valor mínimo, *t* é a iteração atual, t_{max} a quantidade máxima de iterações, *v* o vetor de velocidade, *x* o vetor de posições, C_1 e C_2 constantes positivas comumente referidas como constantes de aceleração, $r1$ e r2 números reais gerados aleatoriamente no intervalo de 0 a 1, y_{ij} a melhor posição encontrada pela partícula e $\hat{y}_j$ a melhor posição global.

O valor das constantes de aceleração foi definido como $C_1 = C_2 = 2$, seguindo a adoção desse valor por diversos trabalhos da literatura (Bansal et al., 2011; Xin et al., 2009; Zhao & Li, 2020). O tamanho do enxame foi definido com base na análise realizada por Piotrowski et al. (2020) de diversos trabalhos da literatura. Eles observaram que os criadores do PSO, autores das variações do algoritmo e pesquisadores utilizaram tamanho do enxame de 20 em seus experimentos de sucesso. Os valores dos parâmetros do PSO estão presentes na Tabela 4.

Tabela 4 - Parâmetros do PSO.

Parâmetro	Valor
w_{max}	0,9
w_{min}	0,4
C1	2
C2	2
Tamanho do enxame	20
Iterações	30

Fonte: Elaborada pelos autores.

O hiperparâmetro da rede MLP "função de ativação em camada oculta" representa uma dimensão no vetor de posições da partícula e foi representado pelo intervalo de 1 a 4, com 1 representando a função Sigmoid, 2 a função Tanh, 3 a função Relu e 4 a função Elu. Para o algoritmo otimizador, 1 representa Adam, 2 RMSprop e 3 Adamax. No cálculo de posição dos hiperparâmetros inteiros (função de ativação em camada oculta, algoritmo otimizador e tamanho do mini-lote) os valores foram arredondados para o inteiro mais próximo (Boubaker et al., 2014). Para os parâmetros que possuem apenas dois valores, a

abordagem binária proposta por Kennedy e Eberhart (1997) foi utilizada. No PSO binário a atualização da posição da partícula para a próxima iteração é dada pela Equação 3.

$$x_{ij}(t+1) = \begin{cases} 0 \text{ se } r < S(v_{ij}(t+1)) \\ 1 \text{ se } r \geq S(v_{ij}(t+1)) \end{cases}, \quad (3)$$

em que v_{ij} é a velocidade da partícula, r é um número aleatório entre 0 e 1 e S é a função Sigmoid, definida por

$$S\left(v_{ij}(t+1)\right) = \frac{1}{1+e^{-v_{ij}(t+1)}}. \quad (4)$$

O vetor de velocidades de cada partícula possui seis dimensões e velocidade máxima, em cada dimensão, como sendo metade do valor máximo do intervalo de valores do hiperparâmetro correspondente (Zhao & Qian, 2011). Por exemplo: a velocidade máxima do hiperparâmetro "algoritmo otimizador", representado de 1 a 3 no vetor de posições, é 1,5.

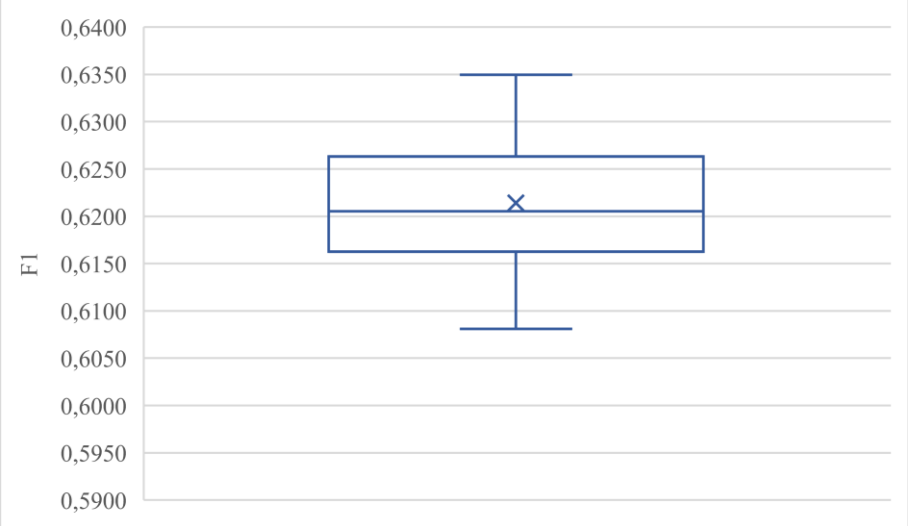
3. Resultados e Discussão

3.1 Algoritmo Genético

Ao fim das 30 execuções, o GA alcançou 15,822 combinações de hiperparâmetros, na qual a melhor combinação resultou em F1 de 0,6349. Considerando o F1 máximo de cada execução do GA, ele alcançou resultados entre 0,6081 e 0,6349, com média de 0,6214 e desvio padrão de 0,0069. O *boxplot* das 30 execuções, apresentado na Figura 1, não possui *outliers* e possui mediana de 0,6205.

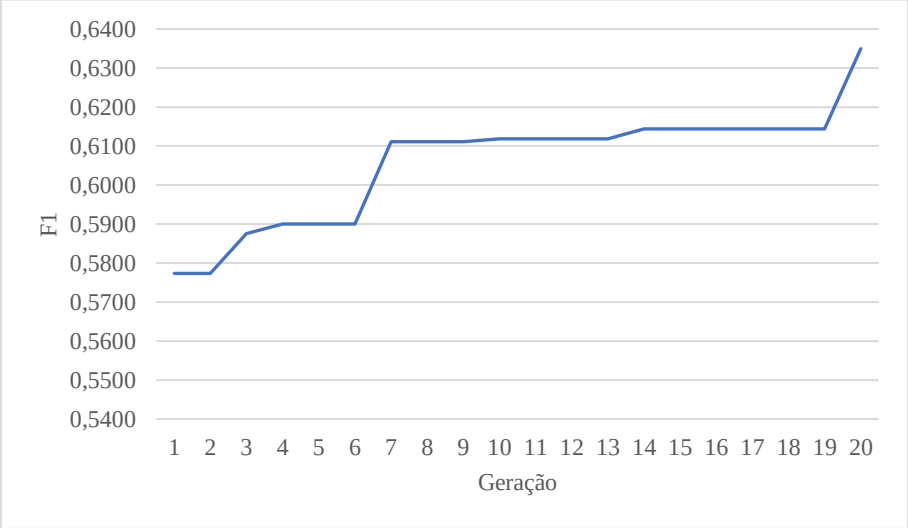
A fim de analisar a execução do GA que resultou na combinação de hiperparâmetros com maior F1, a curva de convergência do GA é apresentada na Figura 2. O GA iniciou com F1 máximo de 0,5773 até alcançar 0,6349 na última geração.

Figura 1 - *Boxplot* das 30 execuções do GA.



Fonte: Elaborada pelos autores.

Figura 2 - Curva de convergência da execução do GA que resultou na melhor combinação de hiperparâmetros.



Fonte: Elaborada pelos autores.

A Tabela 5 apresenta as cinco melhores combinações de hiperparâmetros encontradas pelo GA durante as 30 execuções. Observa-se que o algoritmo otimizador RMSprop está presente em quatro das cinco combinações enquanto o Adamax apenas em uma. O otimizador Adam e a função Elu não estão presentes entre as combinações. A quantidade de neurônios da camada oculta variou de 377 a 857. A taxa de aprendizado ficou compreendida entre 0,005 e 0,035 e o Dropout entre 0,16 e 0,48. O tamanho do mini-lote de 32 aparece em três das cinco combinações, enquanto o tamanho de 64 aparece em duas combinações.

Tabela 5 - Cinco melhores combinações de hiperparâmetros encontradas pelo GA.

Execução do GA	Algoritmo otimizador	Função de ativação oculta	Neurônios ocultos	Taxa de aprendizado	*Dropout*	Tamanho do mini-Lote	F1
18	RMSprop	Relu	857	0,012	0,48	32	0,6349
4	RMSprop	Tanh	650	0,005	0,44	64	0,6344
29	RMSprop	Relu	437	0,012	0,20	32	0,6330
5	RMSprop	Tanh	569	0,006	0,32	64	0,6307
11	Adamax	Sigmoid	377	0,035	0,16	32	0,6278

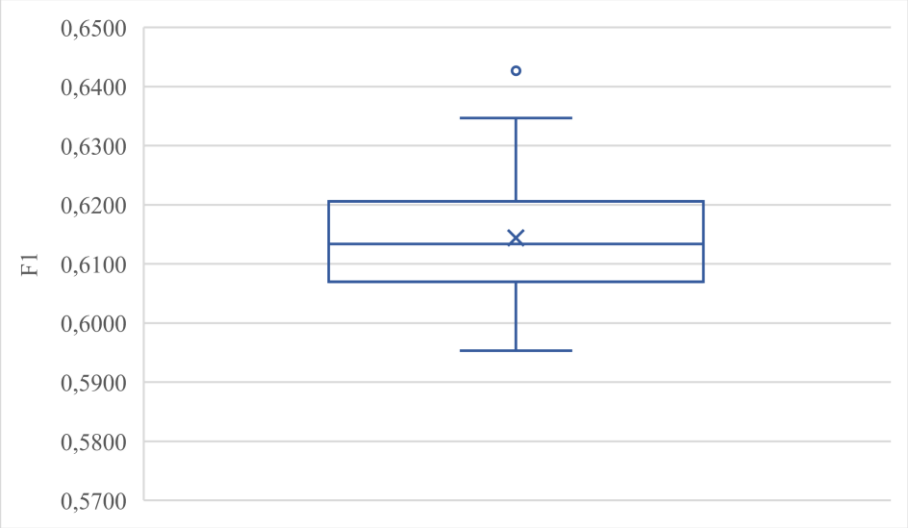
Fonte: Elaborada pelos autores.

3.2 Otimização por Enxame de Partículas

Ao término das 30 execuções, o PSO alcançou 18,000 combinações de hiperparâmetros e F1 máximo de 0,6426 na melhor combinação. O valor de F1 máximo de cada execução do PSO foi inserido em um *boxplot*, apresentado na Figura 3, apresentando valores entre 0,5953 e 0,6426, com média de 0,6144 e mediana de 0,6134. O desvio padrão das 30 execuções é de 0,0115. A execução que apresentou o melhor resultado trata-se de um *outlier* e sua curva de convergência está presente na Figura 4. Da sexta à vigésima iteração o PSO permaneceu em um local mínimo, mantendo F1 máximo de 0,6034, até que na vigésima primeira iteração alcançou o valor máximo de 0,6426.

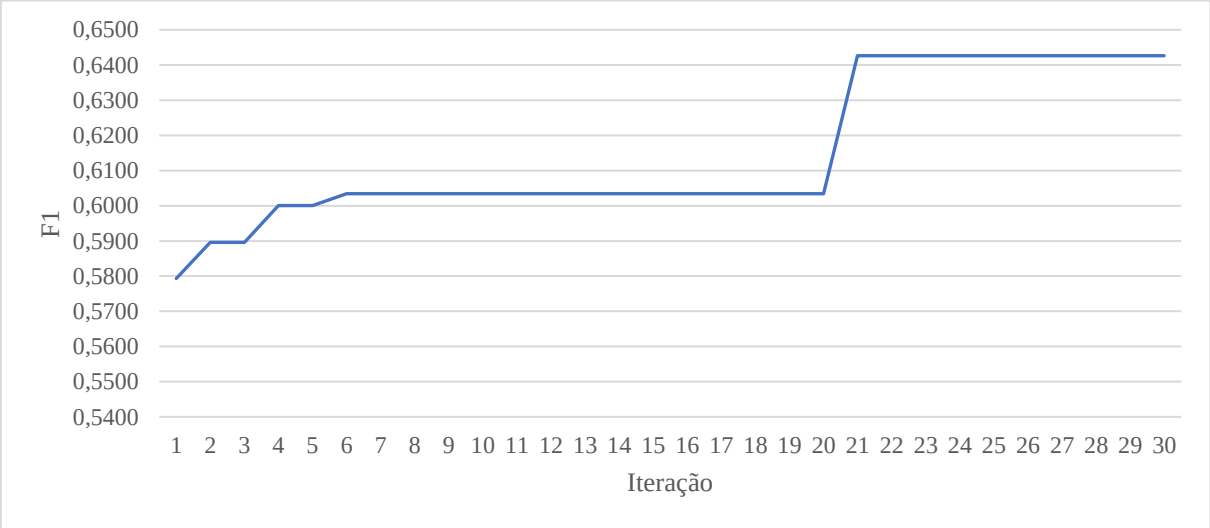
As cinco melhores combinações de hiperparâmetros encontradas pelo PSO são apresentadas na Tabela 6. O algoritmo otimizador Adamax está presente em quatro das cinco combinações enquanto o Adam apenas em uma. O RMSprop não aparece nas combinações. A função Relu está presente em três das cinco combinações enquanto a Tanh aparece em duas. A funções Sigmoid e Elu não estão entre as cinco combinações. O número de neurônios da camada oculta está compreendido entre 319 e 1024 e a taxa de aprendizado apresentou valores entre 0,007 e 0,033.

Figura 3 - *Boxplot* das 30 execuções do PSO.



Fonte: Elaborada pelos autores.

Figura 4 - Curva de convergência da execução do PSO que resultou na melhor combinação de hiperparâmetros.



Fonte: Elaborada pelos autores.

Tabela 6 - Cinco melhores combinações de hiperparâmetros encontradas pelo PSO.

Execução do PSO	Algoritmo otimizador	Função de ativação oculta	Neurônios ocultos	Taxa de aprendizado	*Dropout*	Tamanho do mini-Lote	F1
1	Adamax	Tanh	1024	0,013	0,50	32	0,6426
23	Adam	Tanh	898	0,007	0,45	32	0,6347
18	Adamax	Relu	1024	0,011	0,50	32	0,6323
2	Adamax	Relu	508	0,015	0,50	32	0,6287
11	Adamax	Relu	319	0,033	0,40	32	0,6274

Fonte: Elaborada pelos autores.

4. Conclusão

Neste trabalho, foram utilizados o Algoritmo Genético e o algoritmo de Otimização por Enxame de Partículas para encontrar seis parâmetros da rede neural Perceptron Multicamadas para classificação de requisitos não-funcionais presentes no conjunto de dados PROMISE_exp. O PSO obteve a combinação de hiperparâmetro que resultou no maior F1 (0,6426). Durante

as 30 execuções, o GA encontrou combinações que resultaram em valores de F1 entre 0,6081 e 0,6349 enquanto o PSO alcançou combinações que obtiveram resultados de F1 entre 0,5953 e 0,6426.

Como trabalhos futuros, pretende-se investigar versões modificadas do Algoritmo Genético e do PSO, bem como avaliar o desempenho de outros algoritmos de inteligência computacional, como por exemplo algoritmos de inteligência de enxames, tais como escola de peixes (FSS, *Fish School Search*) e algoritmo do vagalume (FA, *Firefly Algorithm*).

Agradecimentos

Os autores agradecem o incentivo à pesquisa oferecido pelo Programa Institucional de Bolsas de Iniciação Cientifica (PIBIC) e à Fundação Antônio dos Santos Abranches (FASA).

Referências

Abad, Z., Karras, O., Ghazi, P., Glinz, M., Ruhe, G., & Schneider, K. (2017). *What Works Better?* A Study of Classifying Requirements. 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal. https://doi.org/10.1109/RE.2017.36

Bäck, T., Fogel, D. B., & Michalewicz, Z. (1999). *Basic Algorithms and Operators*. IOP Publishing Ltd.

Baker, C., Deng, L., Chakraborty, S., & Dehlinger, J. (2019). *Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks*. 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, USA. https://doi.org/10.1109/COMPSAC.2019.10275

Baluja, S., & Caruana, R. (1995). *Removing the Genetics from the Standard Genetic Algorithm*. Twelfth International Conference on Machine Learning. https://doi.org/10.1016/b978-1-55860-377-6.50014-1

Bansal, J., Singh, P., Saraswat, M., Verma, A., Jadon, S., & Abraham, A. (2011). *Inertia Weight Strategies in Particle Swarm Optimization*. 2011 Third World Congress on Nature and Biologically Inspired Computing, Salamanca, Spain. https://doi.org/10.1109/NaBIC.2011.6089659

Boubaker, S., Djemai, M., Manamanni, N., & M'Sahli, F. (2014). Active Modes and Switching Instants Identification for Linear Switched Systems Based on Discrete Particle Swarm Optimization. *Applied Soft Computing*, 14, 482-488. https://doi.org/10.1016/j.asoc.2013.09.009

Canedo, E. D., & Mendes, B. C. (2020). Software Requirements Classification Using Machine Learning Algorithms. *Entropy*, 22, 1057. https://doi.org/10.3390/e22091057

Carvalho, M., & Ludermir, T. (2007). *Particle Swarm Optimization of Neural Network Architectures and Weights*. 7th International Conference on Hybrid Intelligent Systems (HIS 2007), Kaiserslautern, Germany. https://doi.org/10.1109/HIS.2007.45

Casamayor, A., Godoy, D., & Campo, M. (2010). Identification of Non-Functional Requirements in Textual Specifications: A Semi-Supervised Learning Approach. *Information and Software Technology*, 52, 436-445. https://doi.org/10.1016/j.infsof.2009.10.010

Hoskinson, C. (2011) *Army's Faulty Computer System Hurts Operations*. http://www.politico.com/news/stories/0611/58051.html

Iqbal, T., Elahidoost, P., & Lúcio, L. (2018). *A Bird's Eye View on Requirements Engineering and Machine Learning*. 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan. https://doi.org/10.1109/APSEC.2018.00015

Kennedy, J., & Eberhart, R. (1997). *A Discrete Binary Version of The Particle Swarm Algorithm*. 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Orlando, USA. https://doi.org/10.1109/ICSMC.1997.637339

Kumar, P., Batra, S., & Raman, B. (2021). Deep Neural Network Hyper-Parameter Tuning Through Twofold Genetic Approach. *Soft Computing*, 25, 8747-8771. https://doi.org/10.1007/s00500-021-05770-w

Kurtanović, Z., & Maalej, W. (2017). *Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning*. 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal. https://doi.org/10.1109/RE.2017.82

Lima, M., Valler, V., Costa, E., Lira, F., & Gadelha, B. (2019). *Software Engineering Repositories: Expanding the PROMISE Database*. Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019), New York, USA. https://doi.org/10.1145/3350768.3350776

Navarro-Almanza, R., Juarez-Ramirez, R., & Licea, G. (2017). *Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification*. 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT), Merida, Mexico. https://doi.org/10.1109/CONISOFT.2017.00021

Piotrowski, A., Napiorkowski, J., & Piotrowska, A. (2020). Population Size in Particle Swarm Optimization. *Swarm and Evolutionary Computation*, 58. https://doi.org/10.1016/j.swevo.2020.100718

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation Functions in Neural Networks. *Towards Data Science*, 6(12), 310-316. http://dx.doi.org/10.33564/IJEAST.2020.v04i12.054

Slankas, J., & Williams, L. (2013). *Automated Extraction of Non-Functional Requirements in Available Documentation*. 2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE), San Francisco, USA. https://doi.org/10.1109/NAturaLiSE.2013.6611715

Wolpert, D., & Macready, W. (1997). No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, 1, 67-82. https://doi.org/10.1109/4235.585893

Xin, J., Chen, G., & Hai, Y. (2009). *A Particle Swarm Optimizer with Multi-stage Linearly-Decreasing Inertia Weight*. 2009 International Joint Conference on Computational Sciences and Optimization, Sanya, China. https://doi.org/10.1109/CSO.2009.420

Yoo, J.-H., Yoon, H.-i., Kim, H.-G., Yoon, H.-S., & Han, S.-S. (2019). *Optimization of Hyper-parameter for CNN Model using Genetic Algorithm*. 2019 1st International Conference on Electrical, Control and Instrumentation Engineering (ICECIE), Kuala Lumpur, Malaysia. https://doi.org/10.1109/ICECIE47765.2019.8974762

Zhao, L., & Qian, F. (2011). Tuning The Structure and Parameters of a Neural Network Using Cooperative Binary-Real Particle Swarm Optimization. *Expert Systems with Applications*, 38, 4972-4977. https://doi.org/10.1016/j.eswa.2010.09.154

Zhao, Q., & Li, C. (2020). Two-Stage Multi-Swarm Particle Swarm Optimizer for Unconstrained and Constrained Global Optimization. *IEEE Access*, 8, 124905-124927. https://doi.org/10.1109/ACCESS.2020.3007743