Trabalho IA: Intrusion Detection System Using Optimum-Path Forest

Alexandre Selani[1] and Matheus Rocha[1]

[1]Departamento de Computação, Universidade Estadual Paulista "Julio Mesquita Filho", Brasil

Abstract

Sistemas de Detecção de Intrusão são essenciais para a segurança de redes ao identificar padrões maliciosos no tráfego de dados. Este artigo avalia o desempenho do algoritmo Optimum-Path Forest (OPF) em comparação com Máquinas de Vetores de Suporte (SVM) e K-Vizinhos Mais Próximos (KNN) para a detecção de intrusão, tendo como base a reprodução e comparação com os experimentos do trabalho de referência. Utilizando a base de dados KDD-CUP com três configurações diferentes, os modelos foram avaliados considerando métricas adequadas ao desbalanceamento das classes, como o F1-score e a acurácia estilo OPF, além do tempo de execução. Os resultados indicam que, embora o OPF demonstre desempenho superior na classificação, ele apresenta custo computacional significativamente maior, em comparação com outros modelos. Dessa forma, também discute-se o desafio do desbalanceamento de classes e a relação entre precisão de detecção e eficiência computacional. **Palavras-chave:** Sistemas de Detecção de Intrusão, Optimum-Path Forest, Aprendizado de Máquina, KDD-CUP.

Introdução

Sistemas de detecção de intrusão (Intrusion Detection Systems - IDS) podem ser utilizados em redes corporativas para analisar pacotes de dados e impedir que ataques ameacem a segurança dos dados que por ali trafegam. Devido à grande diversidade dos ataques de rede existentes, algoritmos supervisionados de aprendizado de máquina podem ser úteis no reconhecimento de padrões dentre os pacotes.

Os experimentos realizados nesse trabalho foram inspirados pelo artigo "Intrusion Detection System using Optimum Path Forest" (Pereira et al. 2011), que busca comparar a performance do algoritmo OPF com a SVM-RBF (*Support Vector Machine* com *kernel Radial Basis Function*), SOM (*Self Organized Maps*) e com o classificador Bayesiano treinados sobre conjuntos de dados que descrevem pacotes de dados.

O presente trabalho busca realizar experimentos de forma similar, porém, não idêntica. Os algoritmos testados foram o OPF (*Optimium Path Forest*), a SVM-RBF e o KNN (*K-Nearest Neighbors*).

Materiais e métodos

Essa secção trata dos conjuntos de dados utilizados nos experimentos, da implementação do código, dos métodos utilizados para conduzir o treinamento dos modelos e as métricas utilizadas para avaliar os resultados.

Os experimentos foram suportados por um computador com processador Intel Core i5-10300H, 8 GB de memória RAM e Windows 11.

Conjuntos de dados

Os conjuntos de dados utilizados pelo artigo foram: KDD-CUP e IDS Bag. No entanto, não foi possível encontrar uma cópia do IDS Bag na internet. Por essa razão, apenas o KDD-CUP foi experimentado.

KDD-CUP
O KDD-CUP consiste de amostras de pacotes de dados que podem ou não representar ataques à uma rede. Estão presentes nele amostras de 23 classes (sendo 22 tipos de ataques e 1 classe que representa pacotes comuns) e cada amostra possui 41 features.

Assim como no artigo, foi utilizado 10% do KDD CUP, representando um total de 494021 amostras. As features categóricas foram tratadas com One-Hot Encoder e as duplicatas foram eliminadas do conjunto de dados. A decisão de eliminar as duplicatas favorece a complexidade do treinamento (visto que haverão menos dados) ao mesmo tempo que ajuda a evitar overfitting, uma vez que dados repetidos podem prejudicar algoritmos baseados em distância como o OPF e o KNN.

Foram realizados experimentos utilizando três variantes:

- **KddCup-5:** 6 classes (back DoS,buffer overflow, ftp write, guess passwd, imap e normal);
- **KddCup-10:** 11 classes (ipsweep, land DoS, loadmodule, multihop, neptune, além das classes em KddCup-5)
- **KddCup-15:** 16 classes (nmap, perl, phf, pod DoS, portsweep, além das classes em KddCup-10)

A figura 1 descreve a quantidade de amostras de cada classe em cada variante após o pré processamento. Pode-se notar um grande desbalanceamento entre as classes ao longo de todas as variantes, sendo as classes ”normal” e ”neptune” as mais dominantes. Foi tomada a decisão de não tratar esse desbalanceamento pois a principal métrica utilizada pelo artigo leva em consideração tal característica (mais detalhes na secção ”Métricas utilizadas ”).

Implementação

A implementação dos experimentos foi feita utilizando a linguagem Python, além das bibliotecas Scikit-Learn, utilizada para a implementação da SVM, do KNN, do cálculo de métricas e da validação cruzada; e OPFython, utilizada para implementação do OPF (de Rosa and Papa 2021).

KDD-CUP 5

labels	
b'normal.'	87832
b'back.'	968
b'guess_passwd.'	53
b'buffer_overflow.'	30
b'imap.'	12
b'ftp_write.'	8

KDD-CUP 10

labels	
b'normal.'	87832
b'neptune.'	51820
b'back.'	968
b'ipsweep.'	651
b'guess_passwd.'	53
b'buffer_overflow.'	30
b'land.'	19
b'imap.'	12
b'loadmodule.'	9
b'ftp_write.'	8
b'multihop.'	7

KDD-CUP 15

labels	
b'normal.'	87832
b'neptune.'	51820
b'back.'	968
b'ipsweep.'	651
b'portsweep.'	416
b'pod.'	206
b'nmap.'	158
b'guess_passwd.'	53
b'buffer_overflow.'	30
b'land.'	19
b'imap.'	12
b'loadmodule.'	9
b'ftp_write.'	8
b'multihop.'	7
b'phf.'	4
b'perl.'	3

Figure 1: Contagem de amostras de cada classe para cada variante do KDD-CUP. Fonte: elaboração própria

Treinamento

A fim de obter os resultados, o artigo original executa 10 treinamentos utilizando 50% dos dados para treinamento e o restante para testes e utiliza a média das métricas para representar os resultados. Por restrições de tempo, foi tomada a decisão de executar os treinamentos como descrito a seguir.

Todos os treinamentos se deram da seguinte forma: o conjunto de dados foi dividido em 20 *folds* e ao longo de 10 iterações, um par de *folds* era selecionado aleatoriamente, sendo um utilizado para treino e outro para testes. *Folds* utilizados em uma iteração não eram utilizados novamente. Ao final das iterações, o todo o conjunto de dados terá sido utilizado, sendo 50% para treino e 50% para teste.

A cada iteração, foi realizada uma *5-fold cross validation* com o fold selecionado para treino para otimizar hiperparâmetros antes da fase de treinamento.

Métricas utilizadas

Foram computadas as seguintes métricas: tempo de treinamento, tempo de teste, *F1-score macro*, e acurácia.

Além disso, o artigo original trata o efeito do desbalanceamento de classes nas métricas utilizando um cálculo diferente para a acurácia, o qual será chamado de ”*Accuracy OPF-style*”, denominação

utilizada na biblioteca OPFython. A *Accuracy OPF-style* é calculada da seguinte forma:

Seja Z_2 o conjunto de teste e $NZ_2(i)$ o número de amostras em Z_2 pertencentes à classe i, onde $i = 1, 2, \ldots, c$. A métrica propõe o cálculo de dois termos de erro para cada classe:

$$e_{i,1} = \frac{FP(i)}{|Z_2| - |NZ_2(i)|} \quad \text{e} \quad e_{i,2} = \frac{FN(i)}{|NZ_2(i)|}, \tag{1}$$

onde $FP(i)$ representa os falsos positivos (amostras de outras classes classificadas incorretamente como i) e $FN(i)$ representa os falsos negativos (amostras da classe i classificadas incorretamente como outras classes).

O erro total da classe i é dado pela soma desses dois termos:

$$E(i) = e_{i,1} + e_{i,2}. \tag{2}$$

Finalmente, a acurácia é calculada normalizando a soma dos erros de todas as classes, conforme a equação:

$$Acc = \frac{2c - \sum_{i=1}^{c} E(i)}{2c} = 1 - \frac{\sum_{i=1}^{c} E(i)}{2c}. \tag{3}$$

Resultados e discussão

A tabela 1 descreve os resultados obtidos. As matrizes de confusão para o KDD-CUP 5, KDD-CUP 10 e KDD-CUP 15 estão respectivamente nas figuras 2,3 e 4.

Em geral, percebe-se que a acurácia convencional não permite uma boa análise dos resultados, visto que as classes em maior quantidade nos conjuntos de dados dominam a métrica. Nesse caso, métricas como o F1-score e a Opf-Style Accuracy refletem melhor os resultados, juntamente da analise das matrizes de confusão.

Além disso, pode-se perceber que apesar dos tempos de treino e de teste do OPF serem ordens de magnitude maiores que dos outros algoritmos comparados, sua performance foi ligeiramente melhor. Em questão de custo-benefício, a escolha do algoritmo SVM pode ser a ideal, uma vez que obteve tempos inferiores a 0.5 segundo tanto em treino quanto em teste e resultados melhores que o KNN.

As matrizes de confusão demonstram melhor capacidade do OPF de classificar amostras de classes minoritárias.

Comparação com o artigo original

Os resultados obtidos no artigo original estão descritos na figura 5.

Dataset	Modelos	Acc OPF-Style	Acurácia	F1-Score	Tempo Treino (s)	Tempo Teste (s)
KDD-CUP 5	OPF	0.888 ± 0.078	0.999 ± 0.000	0.699 ± 0.162	33.450 ± 5.898	21.102 ± 4.665
	SVM	0.850 ± 0.086	0.999 ± 0.001	0.640 ± 0.164	0.083 ± 0.069	0.067 ± 0.046
	KNN	0.829 ± 0.084	0.998 ± 0.001	0.583 ± 0.135	0.002 ± 0.000	0.055 ± 0.006
KDD-CUP 10	OPF	0.903 ± 0.026	0.998 ± 0.000	0.684 ± 0.073	92.525 ± 4.083	69.758 ± 6.502
	SVM	0.879 ± 0.036	0.998 ± 0.001	0.652 ± 0.074	0.262 ± 0.093	0.246 ± 0.051
	KNN	0.863 ± 0.030	0.998 ± 0.001	0.601 ± 0.090	0.004 ± 0.000	0.131 ± 0.021
KDD-CUP 15	OPF	0.903 ± 0.030	0.997 ± 0.001	0.697 ± 0.055	94.915 ± 3.932	73.122 ± 6.557
	SVM	0.879 ± 0.043	0.996 ± 0.002	0.667 ± 0.089	0.553 ± 0.295	0.354 ± 0.080
	KNN	0.875 ± 0.036	0.996 ± 0.001	0.633 ± 0.064	0.004 ± 0.000	0.163 ± 0.024

Table 1: Consolidação dos resultados experimentais sobre os datasets KDD-CUP 5, 10 e 15: Comparação de métricas e tempos de execução (Média $\pm$ Desvio Padrão)

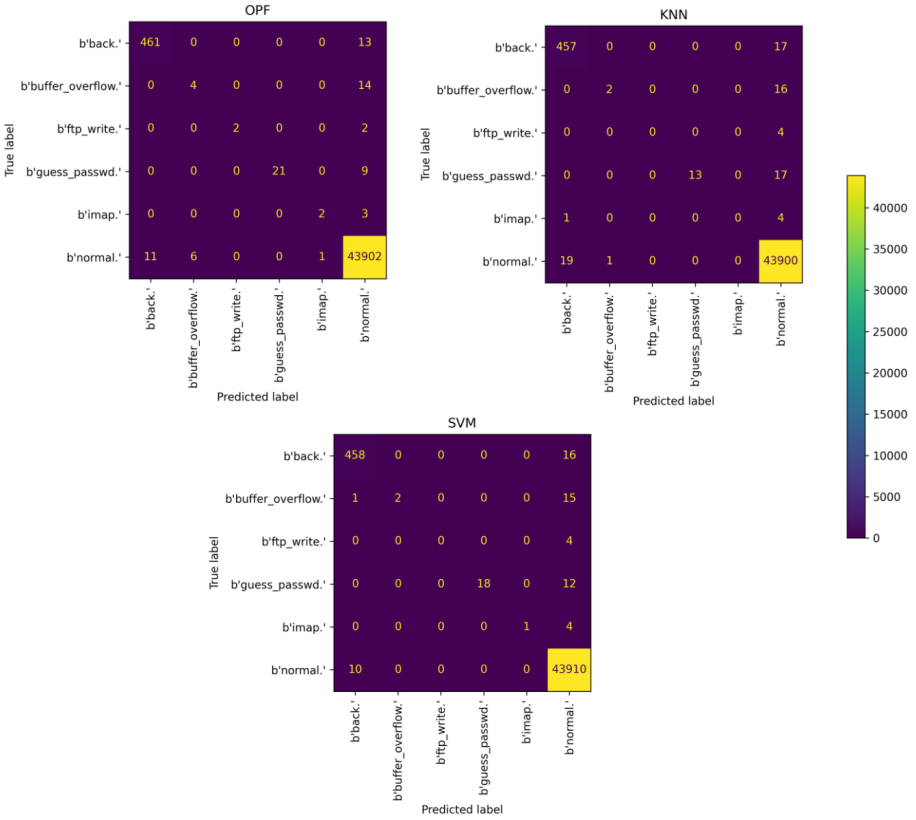

Figure 2: Matriz de confusão obtida nos experimentos com KDDCUP-5. Fonte: elaboração própria

O primeiro ponto a ser abordado são as diferenças nos tempos de treino e teste. Nesse quesito, o artigo original obteve tempos mais longos por algumas razões:

- As implementações utilizadas foram diferentes. Enquanto o artigo utilizou implementações

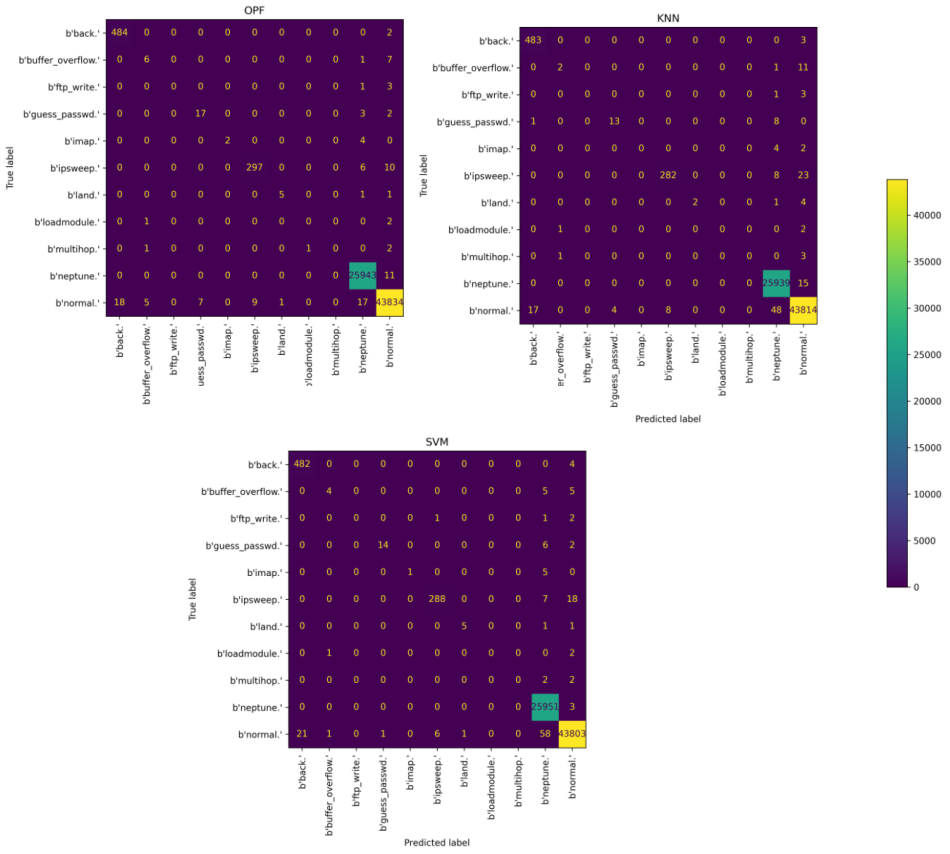

Figure 3: Matriz de confusão obtida nos experimentos com KDDCUP-10. Fonte: elaboração própria

em C (como LibOPF e LibSVM), o presente trabalho utilizou Python nos experimentos.

- A forma como os testes foram conduzidos foi diferente. O artigo original realizou 10 treinamentos utilizando metade do conjunto de dados para treinamento e metade para testes, enquanto o presente trabalho realizou 10 treinamentos utilizando 5% do conjunto para treino e 5% para testes. Portanto, é natural que as médias de tempo dos experimentos do artigo original sejam maiores.

- As máquinas utilizadas foram as diferentes. Apesar de o artigo original não citar as configurações, a publicação já data de mais de 14 anos e espera-se que tenham utilizado um hardware inferior às configurações utilizadas no presente trabalho.

Adicionalmente, os resultados demonstraram diferenças na performance dos classificadores OPF e SVM, principalmente no experimento com o KDD-CUP 5, apresentando resultados melhores que no artigo original. Isso pode ser atribuído à forma como os treinamentos foram conduzidos.

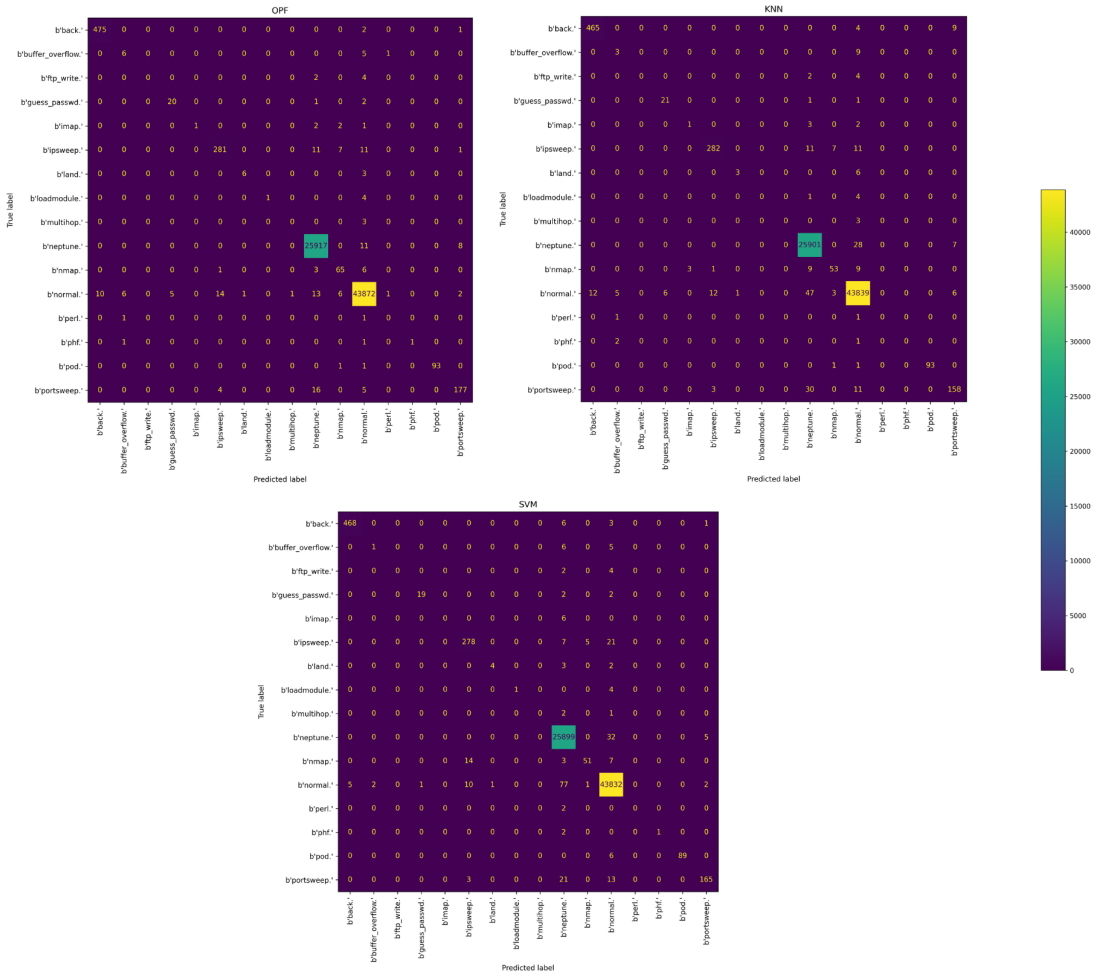

Figure 4: Matriz de confusão obtida nos experimentos com KDDCUP-15. Fonte: elaboração própria

Classifier	Dataset	Acc	Training [s]	Testing [s]
OPF	**KddCup-5**	**80.54±1.66**	**5357.29**	**5083.1900**
Bayes	**KddCup-5**	**80.54±1.66**	**2092.87**	**21964.7300**
SVM-RBF	**KddCup-5**	**81.75±0.86**	**6412.76**	**5014.6600**
SOM	KddCup-5	70.34±4.55	8027.69	24578.00
OPF	**KddCup-10**	**89.31±0.39**	**5568.9798**	**5531.8046**
Bayes	**KddCup-10**	**89.31±0.39**	**2132.3876**	**27407.0000**
SVM-RBF	KddCup-10	85.26±0.12	7035.4401	5348.6321
SOM	KddCup-10	80.73±0.62	9045.0997	27196.865
OPF	**KddCup-15**	**91.49±1.74**	**5970.7832**	**5772.4331**
Bayes	**KddCup-15**	**90.49±1.72**	**2704.8383**	**36475.0000**
SVM-RBF	**KddCup-15**	**92.43±0.64**	**8884.3632**	**6895.0371**
SOM	KddCup-15	85.44±1.81	10513.5331	34614.928

Figure 5: Resultados obtidos pelo artigo: ”Intrusion Detection System using Optimum Path Forest”. O campo ”Acc” refere-se à OPF-Style accuracy. Os classificadores com maior ”Acc” estão em negrito. Fonte: Pereira et al. 2011

Conclusão

Sistemas de detecção de intrusão desempenham uma função crucial na segurança de uma rede: analisar e barrar pacotes de dados que possam representar ameaças.

Esse trabalho buscou experimentar os classificadores OPF, SVM e KNN para tal finalidade, obtendo resultados que apesar de demonstrarem a viabilidade dos modelos, ainda refletem a falha em classificar classes com poucas amostras, o que não é ideal, visto que um único ataque despercebido pode ser devastador. Uma possível forma de melhorar tal aspecto seria coletando mais amostras das classes pouco presentes nos conjuntos de dados, garantindo mais robustez aos modelos.

Por fim, ao comparar os resultados obtidos com trabahos anteriores, foram percebidas algumas diferenças que podem ser atribuídas às condições dos experimentos, que não foram idênticos. Apesar disso, as métricas de performance demonstram certo grau de similaridade nos dois trabalhos.

References

de Rosa, Gustavo H., and João P. Papa. 2021. "OPFython: A Python implementation for Optimum-Path Forest." *Software Impacts,* 100113. ISSN: 2665-9638. https://doi.org/https://doi.org/10.1016/j.simpa.2021.100113.

Pereira, Clayton, Rodrigo Nakamura, João Paulo Papa, and Kelton Costa. 2011. "Intrusion detection system using optimum-path forest." In *2011 IEEE 36th Conference on Local Computer Networks,* 183–186. IEEE.