HIGH SCHOOL EDITION
Journal of Student Research

Evaluating Machine Learning Techniques for Web Robot Detection

Jayan Sirikonda and Mahdieh Zabihimayvan

Farmington High School, USA

ABSTRACT

A web robot, also known as a web crawler, is an automated script that systematically browses the World Wide Web in a methodical manner. Web robots are commonly used by search engines to index web pages, gather information, and update search engine databases. They follow hyperlinks from one web page to another, collecting data and information for various purposes such as indexing, data mining, and website monitoring. Web robot detection is the process of identifying and distinguishing between human users and web robots on websites, which are a major source of web traffic. This process is vital to prevent malicious web robots from having a negative effect on web servers' traffic and their users' privacy. Companies such as Imperva, Inc. use machine learning models to identify malicious bots and prevent them from having unauthorized access to an organization's server. To understand the benefit that machine learning models bring to web robot detection, we used pre-published server log data to construct machine learning models on Orange (a Data Mining Software) and Python that can distinguish between malicious and benign web robots. We evaluated the performance of three well-known machine learning algorithms: kNN, neural network, and decision tree. Based on our experimental results, the neural network gains the highest precision and the lowest false-positive and false-negative percentage of web robots. However, the neural network requires more time to generate the desired output.

Introduction

Web robots are prominent over the Internet, but what exactly are they? A web robot is a program that browses the Internet systematically, following links and extracting data from web pages. Web robots are used by search engines to index content. They can also do tasks such as data mining or monitoring website changes. While there are web robots that operate ethically, there are also malicious variants that aren't ethical, engaging in unauthorized activities such as content scraping without permission.

It is because of these malicious variants that we need the practice of web robot detection, which distinguishes between bots and humans but also distinguishes between malicious and legitimate bots. Without proper web robot detection, these malicious variants can cause chaos by sending spam, gaining unauthorized usage of a user's account, consuming high amount of bandwidth, and much more. By detecting these malicious variants, users will see improved performance, fewer server errors, and increased safety.

To address this problem of web robot detection, we used pre-published data to create 3 different machine learning (ML) techniques (all of which are supervised) in the Orange Data Mining software to learn patterns among the data. Next, to see if we could come up with a better solution using well-known Python packages, we focused solely on creating a neural network with slightly different parameters from the one used in the Orange Data Mining software to achieve a better accuracy rate.

Methods

This section provides a description of the data used in this study and the process we conducted to format and preprocess the data to prepare and clean it before feeding it into a machine learning algorithm. Data preprocessing helps improve the quality of the data, reduces noise and inconsistencies, and enhances the performance and accuracy of machine learning models. Finally, we present the general workflow of the experiments.

Data Description: To collect the data needed, we used Zenodo, a general-purpose open repository, to find a dataset containing server logs from the search engine of the library and information center of the Aristotle University of Thessaloniki, Greece. The server logs in the dataset span an entire month with an average of 131,973 requests per day and a standard deviation of 36,996.7 requests. The server logs are formatted in both JSON and .CSV for functionality.

Data Formatting

I. Orange: The Orange Data Mining software can recognize the Features, Targets, and Metas of a dataset, and it did so rightly for the dataset from Zenodo. Orange assigned 30 columns as Features, one column as Target, and one column as Meta. The Target variable from the dataset was <ROBOT>, which had a value of either 0, meaning a 'benign' web robot, or 1, meaning a 'malicious' web robot.

II. Python: To format the data for Python, we first deleted the <ID> column from the dataset as pandas automatically assigns each row an identifier (ID). We then used the pandas library function .iloc to assign the column <ROBOT> to a variable named target and the rest of the columns to a variable named features.

Data Preprocessing

On the Orange Data Mining software, widgets represent some self-contained functionalities and provide a graphical user interface (GUI). When approaching the dataset, we used two different preprocess methods for imputing missing values: Average/Most frequent and Remove rows with missing values (Figure 1).

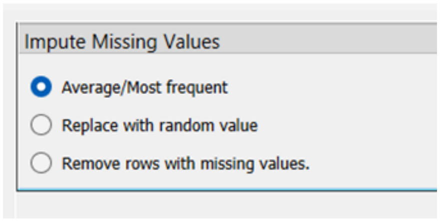
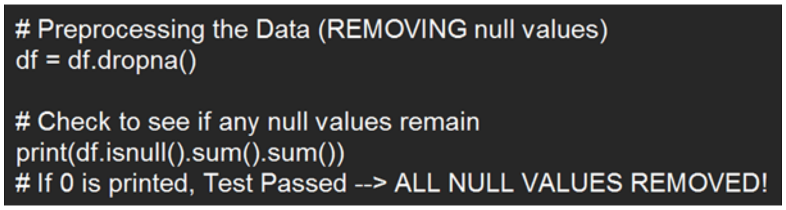

Impute Missing Values
[✓ Average/Most frequent]
Replace with random value
Remove rows with missing values.

Figure 1. Orange Data Preprocess Widget for Imputing Missing Values

To preprocess the data when working with Python, unlike having two different preprocess methods as for Orange, we only used the preprocess of removing rows with missing/null values. Figure 2 shows the Python code written to preprocess the data.


```
# Preprocessing the Data (REMOVING null values)
df = df.dropna()

# Check to see if any null values remain
print(df.isnull().sum().sum())
# If 0 is printed, Test Passed --> ALL NULL VALUES REMOVED!
```

Figure 2. Python code for Preprocessing Data

Workflow

After the data was preprocessed respectively, the data is sampled using the Data Sampler widget in Orange, which uses 80% of the data as train data and the remaining 20% as test data. After dividing the data, we use the following classification models:

Decision Tree Model: This machine learning technique derived its name from trees as it splits data into multiple nodes based on the information gain [8] for categorical target variables (i.e. <ROBOT>). By recursively dividing the dataset, the tree model creates a hierarchical structure, capturing patterns within the data.

Neural Network: We used Multi-Layer Perceptron, a fully connected feedforward artificial neural network [7], that has 100 neurons distributed unequally among hidden layers to extract intricate patterns and relationships within the data, enabling robust learning and prediction.

- K Nearest Neighbors: This classifier (also known as kNN) uses feature similarity for prediction. kNN works by plotting train data on a graph as points and when a new point is graphed, a proximity distance is calculated using different methods such as Minkowski Distance and Manhattan Distance to classify the new point [6].

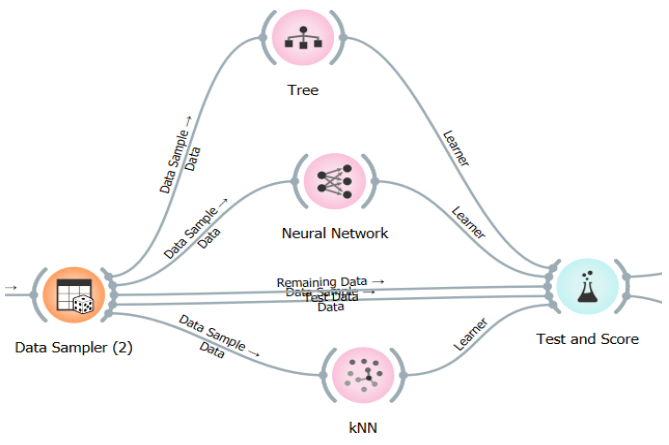
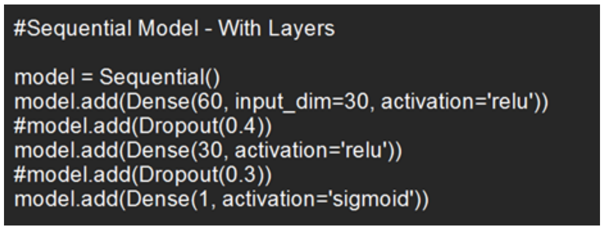


Figure 3. Example of Orange Workflow

Figure 3 shows the Orange workflow of creating the models. These three models are then connected to a Test and Score widget, which outputs a table of evaluation results after conducting various tests on these models with the test data.

Unlike Orange, we only used a neural network when coding in Python (Figure 4). To maintain consistency with our previous workflow in Orange, we divided the data into 80% Train and 20% Test set. After splitting the data, we built a sequential model with three layers totaling 121 neurons. Figure 4 shows code in Python to configure our model.



```
#Sequential Model - With Layers

model = Sequential()
model.add(Dense(60, input_dim=30, activation='relu'))
#model.add(Dropout(0.4))
model.add(Dense(30, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

Figure 4. Code for creating a Neural Network

While writing the Python code, we considered two approaches regarding the appropriateness of drop-out functionality. Dropout deactivates a certain percentage of neurons, forcing the neural network to find more robust relationships within the data [9]. To see the supposed benefit that dropout brings to model accuracy, we saved two versions of the neural network model (one with dropout and one without).

Results

After conducting the tests, the following results were obtained.

Orange

I. Identifying benign web robots: The evaluation table seen in Table 1 is what is outputted by the Test & Score widget. There are various performance statistics here, but we will use the following:
   - Precision: Precision is the proportion of true positives among all instances classified as positive, e.g. the proportion of benign web robots (<ROBOT> value of 0) correctly identified as benign web robots [10].
   - Logistic Loss: Logistic Loss, more commonly known as Log Loss, is a test to see how close the prediction probability is to the actual value, penalizing inaccurate predictions with higher values. If the Log Loss value is close to zero, this indicates better model performance.

When removing null values, the neural network model had the highest precision, edging over the decision tree model by 0.002 and over the kNN by 0.051. The log loss values are also in agreement here: 0.047 is the lowest log loss value and it corresponds with the neural network model, proving this model to be better than the other two when null values are removed.

Table 1. Confusion Matrices of Best Models according to Logistic Loss

Removing Null Values -- Neural Network Model 1						
		Predicted			% False Negatives	% False Positives
		0	1	Total	0.84%	0.46%
Actual	0	9501	49	9550		
	1	89	950	1039		
	Total	9590	999	10589		

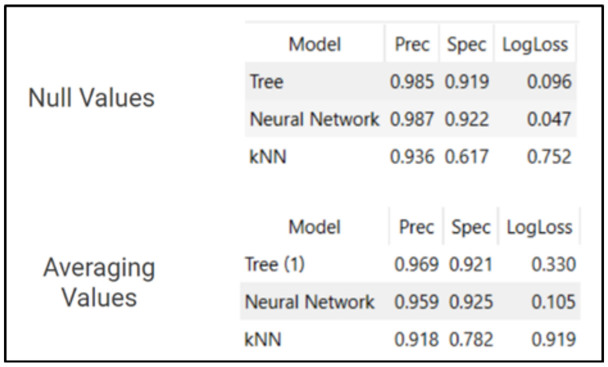
Averaging Values to Replace Missing Values -- Neural Network Model 2

		Predicted			% False Negatives	% False Positives
		0	1	Total		
Actual	0	10466	332	10798	1.70%	2.50%
	1	231	2441	2672		
	Total	10697	2773	13470		

However, when averaging values for rows that have missing values, the decision tree's precision is 0.969, which is higher than the other two models. Despite the decision tree having the highest precision within its preprocess, it did not have the lowest log loss value, indicating that the tree model might not be suited for classification systems like the one we are trying to achieve currently.

II. Identifying Malicious Web Robots: We can use an additional performance statistic to determine a better model for identifying malicious web robots.
- Specificity: Specificity is the proportion of true negatives among all negative instances, e.g. the proportion of malicious web robots (<ROBOT> value of 1) correctly identified as malicious web robots.

From Figure 5, we can see that for the preprocess of removing null values, the neural network had the highest specificity, which means that a very small proportion of all negatives were not true negatives, and we can label this very small proportion as false negatives. For the preprocess of averaging values for rows with missing values, the decision tree had the highest precision, but it was the neural network model that had the highest specificity of 0.925.



Null Values

Model	Prec	Spec	LogLoss
Tree	0.985	0.919	0.096
Neural Network	0.987	0.922	0.047
kNN	0.936	0.617	0.752

Averaging Values

Model	Prec	Spec	LogLoss
Tree (1)	0.969	0.921	0.330
Neural Network	0.959	0.925	0.105
kNN	0.918	0.782	0.919

Figure 5. Evaluation of the neural network with regard to null values in the data

Though the decision tree may have higher precision, we more focus on false-negatives and intend to prevent malicious web robots from being classified as benign web robots, making the neural network model preferable.

III. Comparing Neural Networks between Different Preprocess Methods

The neural network model from removing null values will be referred to as Model 1 and the neural network model from the preprocess of averaging values to replace missing values will be referred to as Model 2. When comparing both model's misclassifications to their respective total data size, Model 1 misclassified 0.84% of malicious robots as benign while Model 2 misclassified only 1.70%. Looking at the % false-positives column, meaning a benign web robot was misclassified as a malicious web robot, Model 1 had 0.46%, which is lower than that of the 2.50% of Model 2. Since Model 1 had fewer false-negatives and false-positives, it is the better of the two models.

Python

We analyzed the performance of the model in Python in two conditions: dropout applied, and no dropout used. The model accuracy with dropout was 80.15% while without dropout was 86.14%, which is 5.99% higher than the model with dropout. These results imply that the Python neural network model performs better in terms of accuracy without using dropout regularization compared with when dropout regularization was applied. We also used the Python model with no dropout to generate the Epoch vs Accuracy overfitting graph. Overfitting is a modeling error in statistics that occurs when a model is closely aligned with its data points, meaning that this model would only work for its initial training data set and will not work well with other data sets. For our model with no dropout, the test accuracy was fluctuating until 7 epochs. After 7 epochs, the accuracy for both the train and test was close 0.9 epochs with slight fluctuations. Though the test accuracy was higher than the train accuracy for the first 7 epochs, the test curve does not go below the train curve, showing that this model didn't have overfitting.

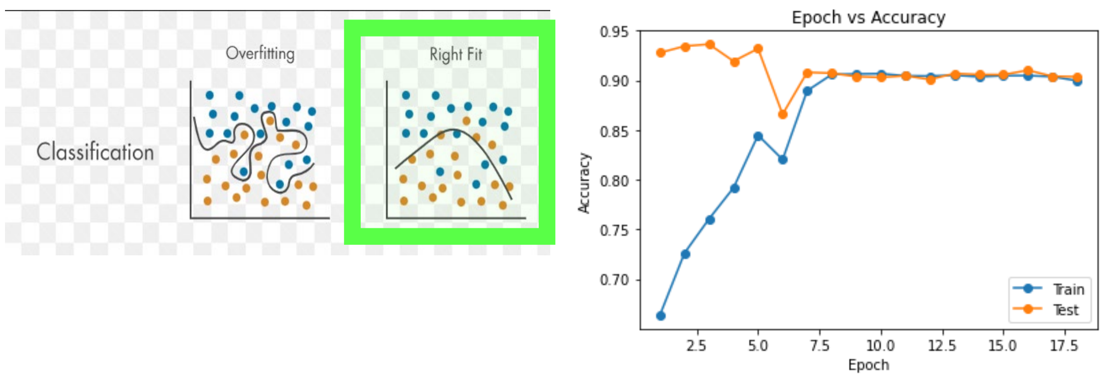


Figure 6. Example of Overfitting vs No Overfitting (left plot). Epoch vs Accuracy Overfitting Graph (right plot)

As indicated in Figure 6, the Python model's logistic loss with no dropout was 0.4099, whereas Orange Model 1's logistic loss was only 0.047. Even though both models are neural networks and were preprocessed by removing null values, the difference in the logistic loss is 0.3629, suggesting that this has to do with how the models were trained.

Conclusion

In this research, we used a decision tree, neural networks, and kNN machine learning models for a measure of how well these models can classify a benign and malicious web robot through two different methods of preprocessing (removing null values and averaging values for missing values within rows of the dataset) over two different platforms (Orange and Python).

When removing null values, the neural network model from Orange proved to be the best as it had the highest accuracy of 98.7% and had a low false-positive and low false-negative rates. The neural network modeled in Python was able to achieve a maximum of 86.14% precision with the same preprocess, which did not top the 98.7% precision. Based on the above analysis, we concluded that ML models can be used to identify web robots and should be used in real-time to protect websites.

Limitations

There are a few limitations. First, the training data was acquired from the server logs of the Aristotle University of Thessaloniki. The server logs could contain data only about robots that were designed to operate within Greece which may not capture the global scale. Another potential consideration is that these web robots captured in the server logs may have only been designed to scrape the university websites, which might not represent the entire proportion of websites on the World Wide Web. Other limitations have to do with how different machine learning models were trained in both Orange and Python. In Orange, the kNN configuration set by default to use k=5 (the number of neighbors), which could be the reason of its low performance as seen in Figure 5.

Future research could focus on alternative methods of parameter tuning for the kNN model to improve its performance and compiling data from multiple websites that span different categories (e.g. banks, companies, library, schools).

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

1 - Brownlee, J. (2020, November 7). How to Identify Overfitting Machine Learning Models in Scikit-Learn. Machine Learning Mastery. Retrieved February 25, 2024, from https://machinelearningmastery.com/overfitting-machine-learning-models/

2 - Gomede, E. (2024, January 6). Understanding the Causes of Overfitting: A Mathematical Perspective. Medium. Retrieved February 25, 2024, from https://medium.com/aimonks/understanding-the-causes-of-overfitting-a-mathematical-perspective-09af234e9ce4

3 - Gross, K. (2020, April 6). Tree-Based Models: How They Work (In Plain English!). Dataiku. Retrieved February 25, 2024, from https://blog.dataiku.com/tree-based-models-how-they-work-in-plain-english

4 - Intuitive Machine Learning. (2020, April 26). K Nearest Neighbors | Intuitive explained | Machine Learning Basics [Video]. Youtube. https://www.youtube.com/watch?v=0p0o5cmgLdE

5 - Lagopoulos, A., & Tsoumakas, G. (2019). Web robot detection - Server logs [Data set]. Zenodo. https://doi.org/10.5281/zenodo.3477932

6 - Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003). KNN Model-Based Approach in Classification. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds) On The Move to Meaningful Internet Systems 2003: CoopIS,

DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39964-3_62

7 - Benardos, P. G., & Vosniakos, G.-C. (2007). Optimizing feedforward artificial neural network architecture. Engineering Applications of Artificial Intelligence, 20(3), 365-382. https://doi.org/10.1016/j.engappai.2006.06.005

8 - Tangirala, S. (2020). Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm. International Journal of Advanced Computer Science and Applications, 11(2). http://dx.doi.org/10.14569/IJACSA.2020.0110277

9 - Srivastava, N., et al. (2014). Dropout: A simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

10 - Handelman, G. S., Kok, H. K., Chandra, R. V., Razavi, A. H., Huang, S., Brooks, M., Lee, M. J., & Asadi, H. (2019). Peering Into the Black Box of Artificial Intelligence: Evaluation Metrics of Machine Learning Methods. AJR. American journal of roentgenology, 212(1), 38–43. https://doi.org/10.2214/AJR.18.20224