MAPREDUCE IMPLEMENTATION FOR MALICIOUS WEBSITES CLASSIFICATION

Md Maminur Islam[1], Subash Poudyal[1] and Kishor Datta Gupta[2]

[1]Department of Computer Science, University of Memphis, USA
[2]Department of Computer Science, Lamar University, Beaumont, USA

ABSTRACT

Due to the rapid growth of the internet, malicious websites [1] have become the cornerstone for internet crime activities. There are lots of existing approaches to detect benign and malicious websites — some of them giving near 99% accuracy. However, effective and efficient detection of malicious websites has now seemed reasonable enough in terms of accuracy, but in terms of processing speed, it is still considered an enormous and costly task because of their qualities and complexities. In this project, We wanted to implement a classifier that would detect benign and malicious websites using network and application features that are available in a data-set from Kaggle, and we will do that using Map Reduce to make the classification speeds faster than the traditional approaches.[2].

1. INTRODUCTION

With the growth of the internet, the number of internet users is increasing tremendously. Most of them are innocent users and they are not aware of the types of internet criminal activities. The users are not conscious enough about the fact that their system or confidential information can be compromised anytime. First attack on computer system by a Cornel graduate student which is known as Morris worm [16] was able to create widespread attention since the worm infected every 1 of 20 computers reducing their computational capacity to fraction compared to their normal capacity and since then, the gravity of maliciousness increased that consequently created cybersecurity as a new field of research and anti-virus created new dimension to software business to most computing environments.

However, since the advent of the internet, the breadth of the internet and the number of users has increased exponentially and most of them are naiveté users along with very few experts. Even the expert users live in constant fear because of the dynamicity of such crimes. Malicious websites are one of the common places to prey the internet users. Hence, providing secure and safe internet to every user is a challenging task and detecting malicious websites is an enormous concern to the security researchers. Lots of existing works going on in this arena. However, we are still far from our goal and lots of scopes to work here which is my motivation to develop a classifier for detecting malicious websites applying different types of machine learning algorithms and improve the performance [18].

2. LITERATURE REVIEW

To combat against malicious websites, several approaches exist which can be divided mainly into two categories: static and dynamic analysis [3]. Static analysis is detecting malicious sites based on some static features such as source code, URL structure, host information etc., Hence, static approaches rely on features that behave differently than benign websites. John P. John et al. [4] proposed a poisoning search results based static approach that are lies on history-based keyword changes in URL and webpages. Birhanu Esheteetal. [5] proposed machine learning algorithms based static approach that extracts features i.e., URL, page source, social reputation etc and

DOI: 10.5121/ijnsa.2019.11503

classifies websites based on those features applying different machine learning algorithms. The Static approach is very efficient and scalable to huge web-pages in cyberspace. However, it has limited success in detecting a foresaid sophisticated malicious sites. On the contrary, dynamic approach esaims to detect malicious websites analyzing runtime behaviors using Client Honeypots [6] or similar techniques [7]. Although, dynamic approaches are very effective but not efficient since they consume lots of resources and needed to run on browsers or operating system which is not feasible. Malicious JavaScript code injection is a very common approach to malicious websites and several considerable types of research have been done in this arena that proposes strategies to detect JavaScript attacks. For example, low and high interaction honeypots [12] propose such strategies to detect JavaScript attacks. On the contrary, many researches concerned about a particular attack type such as drive by downloads [13]. URL classification based on lexical analysis [15] is also a popular approach for URL classification. In fact, URL provides very important features for classification and classification based on those features show considerable performance [15] and such classifier is considered powerful since it takes negligible time and doesn't require to execute the webpages. Google blacklist is also a nice tool to detect malicious sites and has almost 100% accuracy, but it requires continuous updating which may take few hours and hence, infeasible. Kaggle also have some implementation on the dataset that We collected from there and from discussion, We found that they have some models having above 98% accuracy. But there is literary now works done about the efficiency on the perspective of time, Currently 1.8billion websites in this world and it is increasing day by day[19][18]. In computer security, time is the essence. So we must get the result not only accurate also fast as possible.

3. EXPERIMENTAL SETUP

Kaggle is an online community of data scientists owned by Google LLC. that allows users to collect and publish data-sets and work with data scientists and they have lots of competitions to solve data science challenges. They have several data-sets on detecting malicious websites. We downloaded "Malicious and Benign Websites" data-set [8] to classify websites based on application and network features. The tiny dataset contains 1782 datapoints that is enough to run in a single machine. We developed my project in my personal computer that has a core i7 processor and 16GB of RAM. We chose python as my development language and We installed the required packages. The operating system We worked on was windows 10 and Ubuntu 18.

For neural net, we used mlib library (MLlib is Apache Spark's scalable machine learning library.) and for Random forest and Decision tree we used DataFrame library.

For visualization purpose, We had to create service layer and We used python Flask framework to implement the services. Flask is a popular micro framework that doesn't require to install any server to host a website. On the contrary, We used jQuery AJAX to communicate with the python services and used JavaScript D3 visualization tool to show different charts.

4. METHODOLOGY

The first question came to our mind when developing the project is dataset. Kaggle [8] not only a platform for machine learning competitions but also a nice ecosystem for doing and sharing data science. We found the "Malicious and Benign Websites" data-set that is completely compatible with my project.

A typical machine learning model can be described by following block diagram [Fig. 1]. There are several steps in developing a ML model i.e. dataset collection, feature extraction, model

selection, training, prediction and evaluation. Feature extraction includes dealing missing values and transforming features.

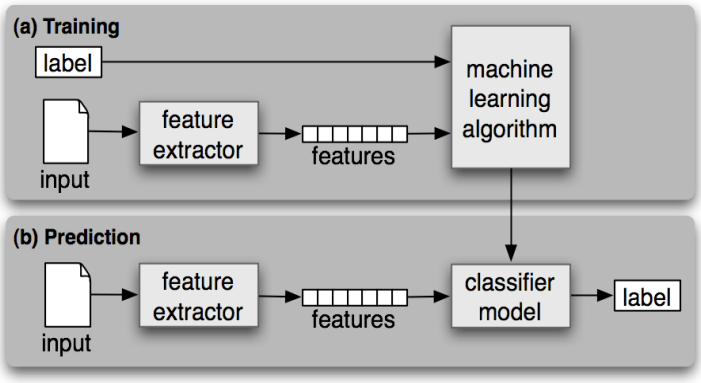

Fig. 1: Block Diagram of ML model

The Kaggle data-set contains 1782 data points and 19 features which has lots of missing values. Hence, the first task is handling the missing values. One possible solution could be to ignore data points having missing values but that would not be a good idea since the dataset is not big enough. Many data points have missing values and hence, discarding those rows would result a tiny training size. So, We followed alternative approach and filled-up missing values with a default value. Secondly, most of the feature values are not in a right format to run different machine learning algorithms. Dealing with categorical features is a challenging task and the performance of various machine learning models mostly depends on dealing with such features.

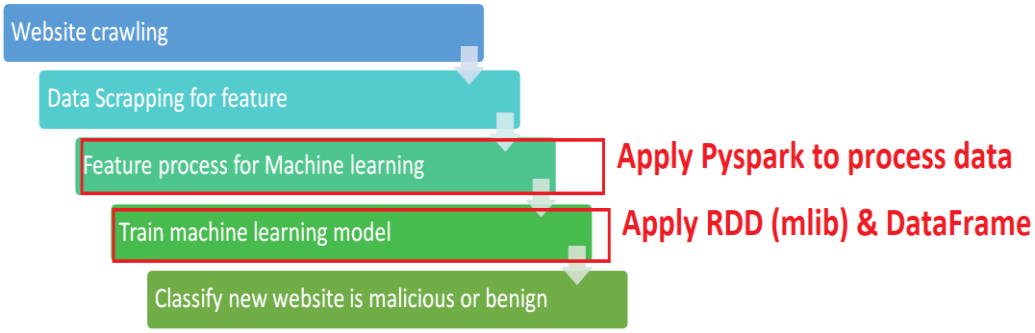

Fig 2: Our Approach combining with big data power

While We started to convert the categorical values to numerical values, We noticed that the categorical features have high cardinality and typical approaches i.e. one hot encoding, dummy binary variables or hashing are not good approaches to deal with such high cardinal categorical features. For example, if we replace the high cardinal categorical features with one-hot encoding, the resultant matrix would be very sparse which might not be good for training ML models. Moreover, the matrix would have too many features and typical machine learning algorithms like, Decision Tree, Support Vector Machine, etc. would not be able to handle too many features.

The encoding algorithms [17] that are based on the correlation between high cardinal categorical attributes and target or class variables provides a nice solution to this problem. The supervised ratio algorithm computes the numerical value for a certain category that is the ratio between the total number of rows that category present in the positive class and total data points.

$$SR_i = \frac{P_i}{N_i + P_i}$$

On the contrary, similar weight of evidence algorithm computes the numerical value for a category according the equation given bellow.

$$WOE_i = \ln\left(\frac{\frac{P_i}{TP}}{\frac{N_i}{TN}}\right)$$

Where,

P_i = number of records with positive class value for the categorical attribute value in dataset.

N_i = number of records with negative class value for the categorical attribute value.

TP = total number of records with positive class value.

TN = total number of records with negative class value.

We replaced the categorical values by both supervised ratio algorithm and weight of evidence algorithm. Hence, Were placed each categorical feature with two columns where one column represents numerical value according to supervised ratio algorithm and another one for the weight of evidence. We wrote it away so we can use the power of Map Reduce provided by pyspark.

The next task is to split the training dataset into two parts, training dataset, and testing dataset. There are several popular practices to divide the dataset. K-fold cross-validation is one of the vastly practiced approaches that We implemented in our project. We used pythons mlib library and data frame library to use the power of MapReduce to decrease the training time of machine learning algorithm.

Then comes the question of model selection where We compared the experiment results among Neural Network (NN), Random Forest, Decision Tree and All the algorithms We selected for comparison are widely used ML models. Our code was uploaded at repository https://github.com/kishordgupta/MapReduce_websiteclassification.

5. EVALUATION

As mentioned earlier, I planned to follow several strategies and compare the results based on accuracy and time. However, accuracy is the most important factor for a good classifier since the dataset is not big. Hence, We mainly emphasis more on accuracy.

The dataset contains imbalanced data [9] which may not reflect the accuracy properly. In my dataset, there are nearly 15% malicious websites that are not enough for evaluation based on accuracy only. Hence, the F1 score which is another measure of accuracy is used to evaluate the classification result.

K-fold cross-validation, sometimes called as rotation estimation or out-of-sample testing is a popular model validation technique to measure how accurately a predictive model performs. In my experiments, We implemented k-fold cross-validation to evaluate the models. However, We also wanted to see the effect of training size. On the accuracy. Underfitting and overfitting is a very trivial case when we try to train various machine learning models. Underfitting means low variance but high bias. When we train a model with less training data, the model may have good

fitting which consequently might show low variance while it may have bad predictive performance having high bias. Typically, there is a trade-off between bias and variance and bias decreases as we increase the training size. However, this is not always true. For example, neural network may show different behavior that usually have high variance with less training data, but variance decreases with increasing training size up to a certain threshold and again variance increases after that training threshold. Hence, We wanted to show how performance changes as we change the training size.

Since various types of loss function have a different impact on the performance of a classifier and performance depends on many other parameters for a certain model as well, We had some experiments where We compared performance changing different parameters. For example, the performance of the neural network may vary as we change the number of hidden layers and the number of neurons in each layer. Additionally, other parameters i,e. the number of maximum iterations may have an effect on performance.

Hence, We compared the model performances for different parameter values and picked the best values to evaluate the models.

As time is considered the deciding factor for evaluating models in my experiments. Although, the dataset selected for malicious websites detection in my project is a very small in size, real-world applications for malicious websites detection might have a very large dataset and if a certain model takes long training or testing time, it might be infeasible for practical applications having good performance in terms of accuracy.

We run our dataset in weka tools, traditional python code, and our developed pyspark codes and compare the time required for training and data processing times of different machine learning model.

6. RESULTS

As mentioned above, We used Neural Network (NN), Decision Tree, Random Forest) to compare the performance

Fig. 3 shows the performance of the models for varying training sizes. Fig. 3 clearly shows that accuracy of Random forest and Decision tree are best, especially the Random Forest that shows 100% accuracy for above 30% training size. On the contrary, Neural Net shows the worst performance even worse than KNN. We are surprised to see a too good performance by both Decision Tree and Random Forest. From my understanding, We think, the algorithms We used to convert categorical values to numerical values had a great impact on the performance of these two classifiers. If we look at the algorithms, we would see that the converted numerical values are another representation of categories and categories best fit in tree type classifiers. On the contrary, Neural Net shows the worst performance because the performance of NN is better for large training data. But our dataset is very small which is not a better fit for NN.

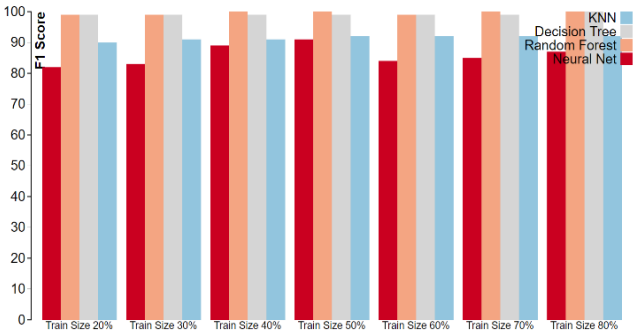

Fig. 3: Comparison of accuracy among different models for varying training sizes.

Fig. 4 shows the accuracy of varying training sizes. From the figure, this is evident that performance increases as we increase the training size. For Decision Tree and Random Forest, since it reaches the best performance (100%) for training size 30%, there is no scope for improvement. But NN and KNN show increasing performance for increasing training data.

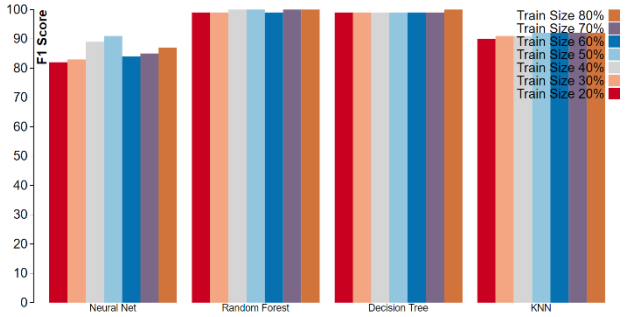

Fig. 4: Comparison of accuracy among different models for varying training sizes.

Table 1 shows the total time (training + testing) comparison for models with varying training sizes. Table 1 clearly shows that larger training makes the models slow which is expected but opposite scenario fir KNN which is also intuitive. Since, there is not much pre-training or preprocessing for KNN and prediction calculation takes place at runtime, more testing data means more calculation which makes the model slow.

Table 1: Time comparison among models for varying training size

Training Size	Neural Net time(ms)	Random Forest time(ms)	Decision Tree time(ms)	KNN time(ms)
80%	6.25	.32	.15	.21
70%	5.2	.28	.13	.26
60%	6.02	.26	.12	.29
50%	5.7	.27	.11	.30
40%	6.17	.28	.10	.32
30%	3.71	.26	.11	.35
20%	2.8	.24	.09	.36

We tried to compare performances of SVM with other models as well but have found that SVM shows almost similar performance to neural network but takes long time to train which would make the graph rendering slow, hence We didn't go with SVM.

In figure 5 we can see that training time on a single node is way more than the traditional time. Weka takes the shortest time, and our process is taking the most significant amount of time, The reason behind this is we are running in a single node. So the power of big data computing is not used here.

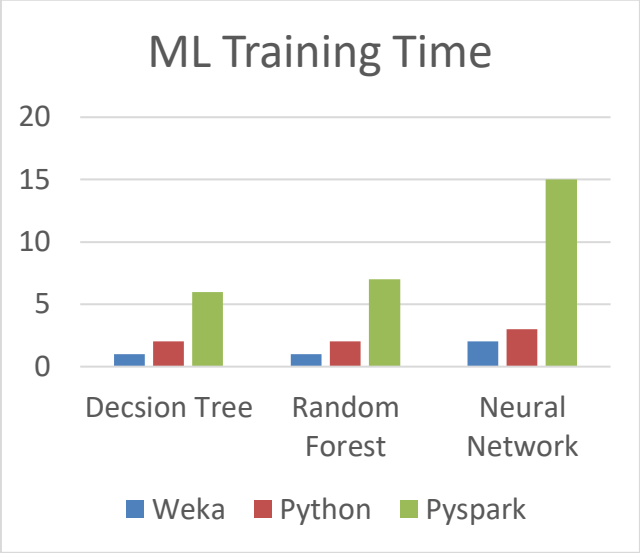

Fig. 5: Comparison of training time among different models for 1 node .

In fig 6 we can see that when we are using double node result started to improve, we are confident if we can run it on more nodes it will be significantly lower time consuming than the traditional approaches.

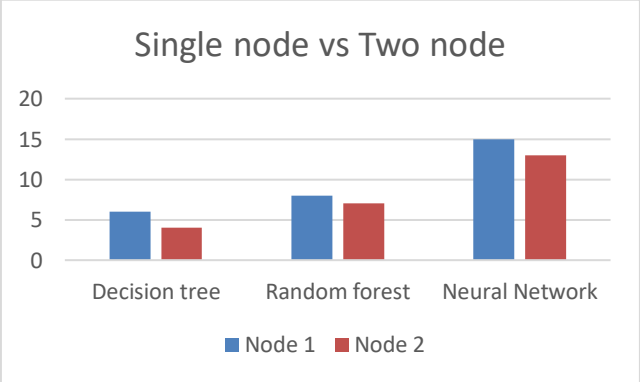

Fig 6: single node vs Two node training time comparison

We also compare the result of time efficiency between iterative programming and our pyspark parallel computing, Due to running in the only 2node result is not satisfactory, But it's evident if we increase the number of clusters, parallel computing will reduce the data process time in the higher margin.

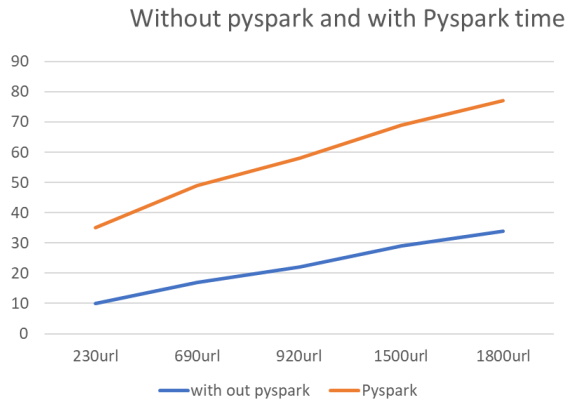

Fig 7: Time comparison between pyspark and without pyspark for data processing.

7. THREAT TO VALIDITY

As we aren't able to run in clusters our results are not concluding, in future, we will try to run in a large group with more significant data set to provide a final decision on time effectiveness improvement by using pyspark.

8. FUTURE WORK AND CONCLUSION

As We already mentioned in the experiment section, the conversion of categorical values to numerical values might have a great impact on the performance of Decision Tree and Random Forest. Hence, in future, We plan to implement another experiment where We compare accuracy with transformation and without transformation of categorical features and see whether my assumption, the transformation had any effect on accuracy. Also, We want to experiment on other big datasets from other sources and compare the performance. The performances of Decision Tree and Random Forest are impressive, and We assume that, those datasets would have different results. But if this is not the case, We are curious to dig more into the issue.

On the performance side the experiment results are outstanding, especially the Decision Tree and Random Forest. However, Lexical Analyzer based classifier that works on features from URL only shows above 98% accuracy. Hence, the outcome of the experiments possibly not unexpected. But in terms of time efficiency, we weren't able to use the full potential of pyspark due to our experimental environment but we can see the sign that parallel computing can give a big boost in time effectiveness.

REFERENCES

[1] https://us.norton.com/internetsecurity-malware-what-are-malicious- websites.html.

[2] https://www.kaggle.com.

[3] https://technical.nttsecurity.com/post/102efk4/detecting-malware-through- static-and-dynamic-techniques.

[4] Eshete, Birhanu. (2013). Effective analysis, characterization, and detection of malicious web pages. Proceedings of the 22nd International Conference on World Wide Web(ACM).

[5] Seifert, Christian & Welch, Ian &Komisarczuk, Peter. (2007). HoneyC - The Low-Interaction Client Honeypot. Proceedings of The IEEE -PIEEE.

[6] Li, Zhou & Zhang, Kehuan&Xie, Yinglian& Yu, Fang & Wang, XiaoFeng. (2012). Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising. CCS

[7] https://www.kaggle.com/xwolf12/malicious-and-benign-websites

[8] https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learning-d43d6fa19d2.

[9] http://leitang.net/papers/ency-cross-validation.pdf

[10] Refaeilzadeh, P., Tang, L., & Liu, H. Cross-validation (2008). URL: http://www. public. asu. edu/~ ltang9/papers/ency-cross-validation. pdf.Hyunsang Choi, Bin B. Zhu, Heejo Lee, Detecting Malicious Web Links and Identifying Their Attack Types, USENIX 2011.

[11] Alosefer, Y., & Rana, O. (2010, April). Honeyware: a web-based low interaction client honeypot. In 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (pp. 410-417). IEEE..

[12] Egele M, Kirda E, Kruegel C. Mitigating drive-by-download attacks: challenges and open problems. IFIPWG 11.4 International Workshop, Zurich, Switzerland,April 2009; pp. 52–62.

[13] Ratanaworabhan, P., Livshits, V. B., & Zorn, B. G. (2009, August). NOZZLE: A Defense Against Heap-spraying Code Injection Attacks. In USENIX Security Symposium (pp. 169-186)..

[14] Yusof, A. R. A., Udzir, N. I., & Selamat, A. (2019). Systematic literature review and taxonomy for DDoS attack detection and prediction. International Journal of Digital Enterprise Technology, 1(3), 292-315..

[15] Hilarie Orman. The morris worm: a fifteen-year perspective. IEEE Security &Privacy, 1(5):35–43, 2003.

[16] https://www.kdnuggets.com/2016/08/include-high-cardinality-attributes-predictive-model.html.

[17] Sen, S., Gupta, K. D., & Ahsan, M. M. (2020). Leveraging Machine Learning Approach to Setup Software-Defined Network (SDN) Controller Rules During DDoS Attack. In Proceedings of International Joint Conference on Computational Intelligence (pp. 49-60). Springer, Singapore.

[18] Dasgupta, D., Shrein, J. M., & Gupta, K. D. (2019). A survey of blockchain from security perspective. Journal of Banking and Financial Technology, 3(1), 1-17.