Domain Generation Algorithms, Malware and Threat Analytics

Jayant Gupta
University of Washington, Tacoma
jayantg@uw.edu

Akhila Bhattarahalli S
University of Washington, Tacoma
akhilabs@uw.edu

Swathika Balasundaram
University of Washington, Tacoma
swathi@uw.edu

1. INTRODUCTION

Botnets are groups of malware-compromised machines or bots, which can be remotely controlled by an attacker using a Command and Control (*C&C*) communication channel. These Bots query certain predefined C&C domain names that resolve to the IP address of the C&C server from which commands will be received. If these centralized C&C server is taken down, the bot master loses control over the entire botnet. To overcome this problem, each bot periodically executes a Domain Generation Algorithm (DGA), which produces a list of candidate C&C domains. The bot attempts to resolve these domain names by sending DNS queries until one of the domains resolves to the IP address of a C&C server.

Botnets have become the main platform for cyber-criminals to send spam, steal private information, host phishing web pages, etc. [9] In order to lessen the risk created by these large numbers of malicious domains, it is important to automatically detect DGA generated URLs.

In this project, we applied different machine learning methods to develop a classifier that can distinguish DGA generated URLs from clean domain names, as well as a classifier that can identify which particular family of DGAs has generated a malicious URL.

2. EXPERIMENTAL SETUP

In this section we would describe the experimental setup, describing the dataset, pruned dataset, features and the baselines used to assess the system.

2.1 Dataset

Table 1 lists the distribution of classes in the dataset. Overall the dataset contained 1137036 distinct URLs. The given dataset is highly skewed in favor of NON-DGA class having 900,000 (79.15%) distinct URLs. To reduce the skewness of the data we reduced the data, reducing the number of DGA URLs; so that, less frequent class could be represented easily. Section 2.2 describes the creation of filtered dataset.

Bamital	Clicker	Conficker A	Conficker B	Conficker C
6529	37	20925	20925	45000
CoreFlood	Cryptolocker	DarkComet	Darkbot	Expiro
55	83700	45	81	327
ExpiroZ	Flame	Flashback	InfoStealerShiz	Kelihos
20	76	419	1800	340
Neverquest	NewZeuSGameOver	Pushdo	Ramdo	Ramnit
5	882	7398	4549	910
Ransomware	RunForrestRun	Shiz	Sinowal	TinyBanker
15	672	900	2236	25
UrlZone	Virut	ZeroAccess	ZeusGameOver	NON-DGA
27	8370	168	30600	900000

Table 1: Distribution of classes in the input dataset

2.2 Pruned Dataset

We extracted rows from original data to get Pruned Data. Separate Pruned Data for Binary and Multi-Class training was created. The Binary pruned Data has equal instances of DGA (237036) and Non-DGA (237036). The Multi-Class pruned Data contains 237036 dga URLs and 90000 non-dga URLs. The non-dga URLs were randomly sampled (10% out of total) from the given non-dga URLs.

Both Multi-Class and Binary are further divided into train (80%) and Test (20%). Table 2 tabulates the number of rows for each train and test data.

	Train	Test
Binary	379258	94814
Multi-Class	261629	65407

Table 2: Train and Test data for Binary and Multi-Class classification

2.3 Features

To extract feature we considered various parts of the URL using $L-i$ notation. For example consider *www.example.com*, L-1 is com, L-2 is example.com and L-3 is www.example.com. Some features are extracted for various individual parts.

We considered three key properties of a URL. First, entropy of the URL, it indicates the lexical randomness of the characters used. Entropy of each URL is calculated as follows:

$$H(URL) = \sum_{c \epsilon URL} p(c) log(\frac{1}{p(c)})$$

We calculated H_1, H_2 and H_3 as the entropy of L-1, L-2 and L-3 components of the URL respectively. Second, we considered N-Gram distribution features in the URL. We calculate Mean, Median and Standard Deviation of the URL character distribution. We believe that such features make more sense when considering a set of URLs instead of a single URL. However, in our experiments, some of these features came up as relevant features. Third, we consider structural Features of the URL such as, Number of dots, length, Number of distinct characters and the suffix of the URL. Table 2.3, lists all the features, that were calculated for each of the URL.

Entropy Based Features	
$H_{1,2,3}$	Entropy of $URL_{1,2,3}$
N-Gram Distribution Features	
$\mu_{1,2,3,4}$	Mean of n-gram frequencies
$Me_{1,2,3,4}$	Median of n-gram frequencies
$\sigma_{1,2,3,4}$	Standard Deviation of n-gram frequencies
Structural Features	
#dots	Number of Dots
URL length	Length of the URL
n	Number of distinct characters
suffix	URL suffix

Table 3: List of extracted Features

2.4 Baseline

To assess our results we created two baselines as follows:

1. Sampling based on Prior Probabilities: Prior probabilities were calculated using the training data. Then, for each test data input, class was chosen from this prior distribution.

2. Majority Class: Majority class is determined from the training data. Since, the data was skewed in favor of Non-DGA URLs, all the test URLs were classified as Non-DGA in this baseline

   It can be noted that URL features are not needed for the aforementioned baselines. Table 4 shows the baseline accuracy for the binary and multi-class classification.

	Binary	Multi-Class
Baseline1	50.02%	17.98%
Baseline2	50%	27.92%

Table 4: Classification Accuracy for Baselines

2.5 Tools Used

- **AzureML:** A cloud based platform from Microsoft that consists machine learning capabilities. [1]
- **R:** It is a free software environment for statistical computing and graphics. [7]
- **Python:** Python is a programming language that lets you work quickly and integrate systems more effectively. [5]

3. METHODOLOGY

For this work, we took train, test and analyzed methodology, wherein, we have tried to build different types of classifiers, both binary and multi-class. In the end we chose the classifier that gave the best results on the training data. For each classifier we have changed the key parameters to observe the change in accuracy and to understand the importance of the parameter.

3.1 Decision Tree

Classification and regression trees are machine learning methods for constructing prediction models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition. As a result, the partitioning is done based on the selected variable. Selection of variable depends on the Information gain measure of each variable. Usually, the variable with the highest information gain is selected.

Classification trees are designed for dependent variables that take a finite number of unordered values, with prediction error measured in terms of misclassification cost. Regression trees are for dependent variables that take continuous or ordered discrete values, with prediction error typically measured by the squared difference between the observed and predicted values. [10]

Binary Classifier using decision trees was built in R and the following parameters were used.

- Minsplit : The minimum number of observations that must exist in a node in order for a split to be attempted.
- Minbucket : the minimum number of observations in any terminal 'leaf' node.
- Cp : Any split that does not decrease the overall lack of fit by a factor of 'cp' is not attempted.
- Maxdepth : Set the maximum depth of any node of the final tree, with the root node counted as depth 0.
- control : Various parameters that control aspects of the 'rpart' fit.

An accuracy of 72.94% was obtained with parameters having the following values, Minsplit:20, Minbucket:7, Cp:0.01, Maxdepth:30, control:10-fold cross validation. The model had H_2, url_length, n, H_1, mu_1 as the **best features**. Importance of feature is stored in the R object of the model, it is determined by information gain of the variable.

Observation: We performed the same experiments without using 10-fold cross validation and the trained model had an accuracy of 67.85%. We obtained the same set of best features, H_2, n, url_length, H_1, mu_1, except the order of distinct characters and URL length were changed. This implies, that feature selection is a very important characteristic of decision trees.

Multi-Class Classifier using decision trees was built in R and the parameters were same as for binary classifier.

An accuracy of 71.94% was obtained for the parameters with following values, Minsplit:20, Minbucket:7, Cp:0.01, Maxdepth:30 10-fold cross validation. The model had url length, H_2, n, H_1, mu_1, sd_1 as the **best features**. Importance of feature is stored in the R object of the model and is determined by information gain of the variable.

Observation: We performed the same experiments without using 10-fold cross validation and the trained model had a reduced accuracy of 62.16%. Pruning of the tree (trained using 10-fold cross validation) was also tried where the depth of the pruned tree was 23% of the tree, the accuracy obtained was 70.98%. This shows that, pruning of the tree can be very helpful in situations where pruned tree gives comparable accuracy.

3.2 Boosted Decision Tree

Boosting is an ensemble method, which is most powerful in machine learning. In boosting, we maintain a vector of weights for training examples. All training examples will be initialized with uniform weights. We build a model for each training example and pass on to the next iteration. In the next iteration, the weight is increased for the misclassified examples. The next model will be built by giving more weight to the misclassified examples. This process is continued until the model does no good than the current model. Finally, these models are combined by weighted voting.

Binary Classifier was built using boosted decision tree in AzureML, having the following **parameters**.

- Number of trees constructed (nTree)
- Maximum number of leaves per tree (nMaxLeaves)
- Minimum number of samples per leaf node (nMinSample)
- Learning rate (LR)

Accuracy of 85.7% is observed for the following set of parameters. nTree: 300, nMaxLeaves: 20, nMinSamples: 10, LR: 0.2

Observation: Figure 1 shows the variation in accuracy with the increase in number of decision trees. The accuracy improved by varying nTree parameter from 50 to 350. The other parameters did not have effect on the accuracy. The accuracy remains constant after 250 decision trees. This could mean that we have generated right amount of trees to train the model. The running time increased from 2 minutes to 10 minutes with the increase in the number of decision trees. In Azure ML, all features are selected to build the model and then accuracy is observed. After removing string n-gram features and entropy features, the accuracy did not change. This means that these features did not contribute towards the accuracy of the model. However, features such as the number of distinct characters in a URL, number of dots in a domain name, suffix of a URL and URL length turned out to be the **best features**. This is because the removal of any of these features reduced the accuracy of the model.

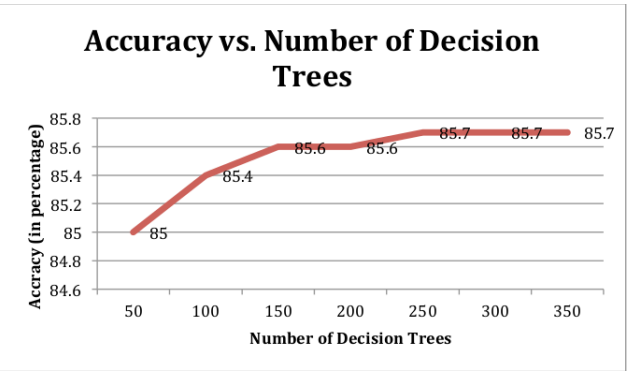

Figure 1: Change in accuracy versus number of decision trees.

3.3 Support Vector Machines (SVM)

Support Vector Machines are supervised learning models used for classification and regression analysis. In SVM, each training example is represented by a point in space. The training algorithm builds a model that assigns new examples into one category or the other. It maps these points so that the examples of separate categories are divided by a clear gap that is as wide as possible. The new examples are mapped to the same space and predicted to belong to a category based on which side of the gap they fall on. [3]

Binary Classifier was built using Support Vector Machines in AzureML, having the following **parameters**.

- Number of iterations (nIterations)
- Kernel-In Azure ML, users do not have the privilege to select the "Kernel" type. It supports only linear type.

Accuracy of 79% is observed for the following value of parameters. nIterations: 800, kernel: Linear.

Observation: The accuracy improved by varying nIterations parameter from 100 to 800. The accuracy remains constant after 600 iterations. This could mean that we have reached the right number of iterations to train the model. The running time increased from 5 minutes to 8 minutes with the increase in the number of iterations.

3.4 Random Forest

Random Forest is an ensemble method. In this method, several decision tree classifiers are built by selecting random set of features for each tree. Each decision tree is trained on the replicas of training dataset and selected features. Each training sample can be chosen many times (approx. 2/3 of the sample data is used to train each tree). The collection of models usually improve the accuracy and performs better than single classifier. For prediction, input vector is assigned a class by each classifier. The final class is chosen to be majority of the predicted classes.

Binary Classifier was built using Random Forest in R, having the following **parameters**.

- Number of trees (nTree).: 30
- Number of variables tried at each split.(nVar): 5

Accuracy of 73% is observed for the following value of parameters. nTree:30, nVar:5. We found that for this model the best features were H_2, H_1, URL_length, $H.$, n, mu_1. Importance of variable is determined by decrease in accuracy over all the classes.

Figure 2 shows the change in accuracy versus the number of decision trees. It can be seen that highest Accuracy occurs when number of trees is 100. Since, there is no significant change in accuracy we have limited the number of trees to 30.

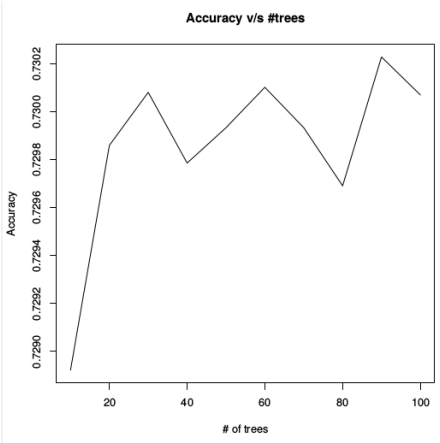

Figure 2: Change in accuracy versus number of decision trees.

Multi Class Classifier was built using Random Forest in R, having **parameters** similar to that of binary classifier. **Accuracy** of 72% is observed for the following value of parameters. nTree:50, nVar: 5. The **Best Features:** for this model were URL length, H_1, H_2, n, mu_1, sd_1, X.dots.

Observation Accuracy of 72.05% occurs when number of trees is 80. From Figure 3 we see that, no significant change is observed when we increase the number of trees we have limited the number of trees to 50. For both binary and multiclass classifiers we observed that increase in the number of trees does not result in much significant change. This shows that, the lower number of trees are sufficient to capture the necessary class patterns, used to classify any given URL.

3.5 Decision Forest

Decision forest is a modification of original Random forest algorithm designed by Leo Breiman and Adele Cutler. This algorithm is based on the use of decision trees to get result by uniform voting and the training process randomization [6]. In this method, we build several models by generating random set of features. We generate replicas of training set by sampling with replacement. This means that the same training example may appear multiple times in the replicas. We build the model for each replicate. We combine the models by uniform voting and hence the majority class

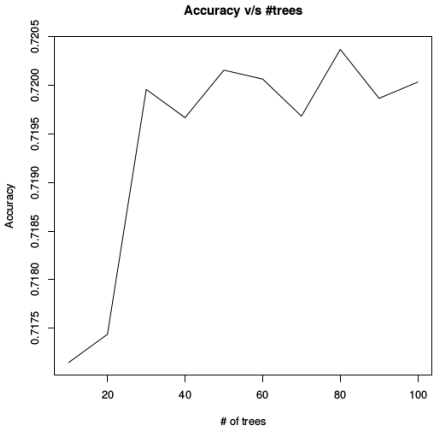

Figure 3: Change in accuracy versus number of decision trees.

wins. This typically improves the accuracy. In decision forest, multiple decision trees are built using the training set. The uniform voting of all models is considered to get the final model. **Binary Classifier:** was built using Decision Forest in AzureML, having the following **parameters**

- Re-sampling method (reSampling)
- Number of Decision Trees (nTrees)
- Maximum depth of the decision trees (maxDepth)
- Number of random splits per node (nSplits)
- Minimum number of samples per leaf node (nMinLeaves)

Accuracy of 80.5% is obtained for the following set of parameters. reSampling: Bagging, nTrees: 350, maxDepth: 700, nSplits: 128, nMinLeaves: 1.

# Decision Trees	Maximum Depth	Accuracy (%)
50	300	78.3
100	500	78.9
150	500	79.3
200	600	79.8
250	700	80.2
300	700	80.4
350	700	80.5

Table 5: Variation in accuracy with the increase in number of decision trees and maximum depth of decision trees.

Observation: Table 5 shows that the accuracy improved by varying nTrees and maxDepth parameters. The variation of other parameters did not affect the accuracy and hence they are kept constant during the experiment. The maxDepth had more influence on the accuracy. If the maximum depth is set to lower values (say 20), an accuracy of less than 20% is observed. The running time increased significantly with the increase in number and maximum depth

of decision trees. It varied from 10 minutes to 45 minutes. In Azure ML, all features are selected to build the model and then accuracy is observed. After removing string n-gram features and entropy features, the accuracy did not change. This means that these features did not contribute towards the accuracy of the model. However, features such as the number of distinct characters in a URL,number of dots in a domain name, suffix of a URL and URL length turned out to be the **best features**. This is because the removal of any of these features reduced the accuracy of the model.

Multi-class Classifier was built using Decision Forests in AzureML having **parameters** similar to the binary classifier. **Accuracy** of 77% is obtained for the following set of parameters. ReSampling: Bagging, nTrees: 250, maxDepth: 700, nSplits: 500, nMinLeaves: 1

Result: Table 6 shows the variation in accuracy with the increase in number of decision trees, maximum depth of decision trees and number of random splits per node.

nTrees	maxDepth	nSplits	Accuracy (%)
10	400	200	76.9
50	700	500	77.26
100	700	500	76.9
150	700	500	77
200	700	500	77
250	700	500	77

Table 6: Variation in accuracy with the increase in number of decision trees and maximum depth of decision trees.

Observation The accuracy improved slightly by varying number, maximum depth of decision trees and number of random splits per node. The other parameters did not affect the accuracy and hence kept constant throughout the experiment. The "nSplit" specifies the number of splits generated per node from which the optimal split is selected. This parameter affected the accuracy significantly. An accuracy of 25% is observed for the value of 128 (default value). This means that model could not predict better with the less number of splits per node. Hence by increasing this parameter, the model generated many splits per node and selected the optimal one, which in turn improved the accuracy. The running time is generally high for multi-class classifiers. It increased significantly with the increase in the values for the above-mentioned parameters. It varied from 45 minutes to 2 hours.

3.6 Neural Network

The Artificial Neural Network is inspired by the biological neural network [2]. This typically depends on large number of inputs and is generally unknown. This model accepts the input and learns the hidden units and generates the output, based on the results at the hidden layer. This model uses the back propagation method to correct the error at the hidden layer and at the output layer. This model typically needs more training time.

Binary Classifier was built using Neural Network in R, having the following **parameters**

- Number of hidden layer (nHiddenLayer)
- Number of hidden nodes (nHiddenNode)
- Number of learning iterations (nIterations)

Accuracy of 78.2% is observed for the following parameters, nHiddenLayer:1, nHiddenNodes:3, nIterations:100.

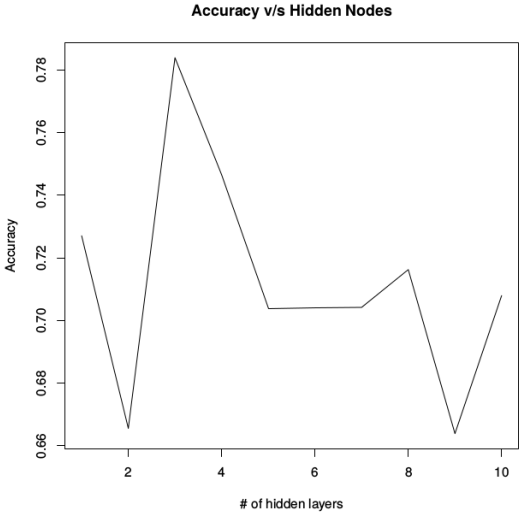

Figure 4: Change in accuracy versus number of hidden nodes

Observation: Figure 4 shows the change in accuracy with increase in the number of hidden nodes. It can be seen that highest accuracy when the number of hidden nodes is 3, on further increase the accuracy goes down. This shows that when sufficient nodes are reached to build the model, new nodes can result in a negative effect to the model.

Multi-class Classifier was built using Neural Network in AzureML, having the following **parameters**

- Number of hidden layer (nHiddenLayer)
- Number of hidden nodes (nHiddenNodes)
- The learning rate (LR)
- Number of learning iterations (nIterations)
- The initial learning weights (initWeights)
- The type of normalizer (normalizer)

Accuracy: An accuracy of 12.4% is observed for the following set of parameters: nHiddenLayer: 1, nHiddenNodes: 30, LR: 0.1, nIterations: 10, initWeights: 0.1, normalizer: Min-max normalizer [1].

Observation: The accuracy of this model is lower than any other models we have tried. The nHiddenNodes parameter

[1]Min-Max normalizer is described in Appendix-I

is varied and other parameters are kept constant during the experiment. The running time of this model is significantly high even with the low values of parameters (1 hidden layer). The running time was around 4 hours for 30 hidden nodes and increased to more than 5 hours for greater than 30 hidden nodes[2].

3.7 NaÏve Bayes:

A Naïve Bayes classifier is a supervised learning method, and it is a probabilistic classifier based on Bayes theorem. It is an independent feature model where the classifier assumes that each feature is independent or unrelated to any other feature within a class. Naïve Bayesian model is easy to build and there is no complicated parameter estimation, which will be easy while working on a larger dataset. There are two steps in Naïve Bayes:

Training Step: Using the training data, the method estimates the parameters of a probability distribution, assuming predictors are conditionally independent given the class.

Prediction step: For any unseen test data, the method computes the posterior probability of that sample belonging to each class. The method then classifies the test data according the largest posterior probability. [8]

Classifier: Binary and Multi class

Parameters Used:

- kernel: If True kernel density estimation is used for density estimation, if False normal density estimation is used.
- FL: It is a parameter used for Laplace correction.

Accuracy: Binary Classifier: 64.2%
Multi-Class Classifier: 58.5%

After 10 fold Cross Validation:

Binary Classifier: 71.23%
Multi-Class Classifier: 60.14%

The above accuracy is obtained using the following parameters:
Kernel = False, because using kernel is more appropriate only for predictors having continuous distribution so we used normal density estimation which is appropriate predictors having normal distribution.
FL = 0, which is a default factor denoting no Laplace correction.

Observation: We applied Naïve Bayes on the complete dataset. We also tried to train the model using 10-fold cross validation to see if there is an improvement in accuracy and we noted a significant improvement in accuracy from 64.2% to 71.23% for binary classifier and 58.5% to 60.14% for multi-class classifier. The training period and memory usage was too high when using full dataset. Time taken to build the model varied from 5 to 6 hours. URL length, suffix, number of distinct characters turned out to be the **best features** for this model, when trying to remove these features the accuracy got reduced.

[2]The accuracy might have improved with the increase in nHiddenLayer and nIterations in addition to the nHiddenNodes. Due to the increase in running time, the experiment is repeated for lower parameter values only.

3.8 Logistic Regression

Logistic Regression is used for predicting the probability of occurrence of an event and is popular for classification tasks. This algorithm will predict the probability of occurrence of an event by fitting the given data to a logistical function. [4]

Binary Classifier was built using logistic regression in Azure ML having the following parameters. **Parameters:**

- Optimization tolerance (OT): It sets a threshold value, if the improvement between iterations is less than the threshold, logistic regression stops and returns the current model.
- L1 regularization weight and L2 regularization weight (L1RW and L2RW): Regularization methods are used to prevent over fitting. Memory size (MemSize): This parameter sets the amount of memory used for storing the computational steps.

Accuracy of 81% was obtained using following parameters: OT:10^{-7}, L1RW: 1, L2RW: 1, MemSize: 20.

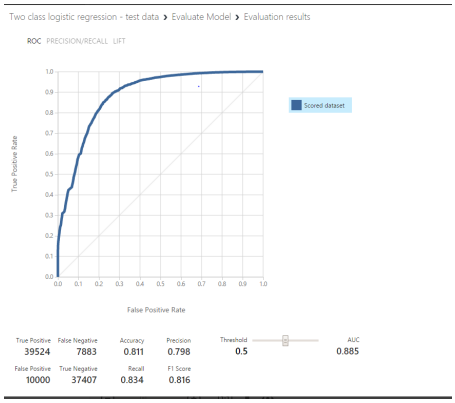

Figure 5: True Positive Rate vs False Positive Rate for Two-Class Logistic regression

In the Figure 3.8 true positive and true negative are the observations which were correctly predicted. And false positive and false negative are errors that are wrongly predicted observations. Accuracy is the ratio of correctly predicted observation. According to the above result accuracy is calculated as,

$$\frac{TP+TN}{Totalnumberof Events} = \frac{76931}{94814} = 0.811$$

Threshold by adjusting the threshold we can decrease the frequency of one type of error at the expense of increasing other error. We have set the threshold value to 0.5.

Methods	Accuracy(%)		Built-in
	Test data	Hidden data	
Baseline I	50.02	50.13	R
Baseline II	50	79.15	R
Naïve Bayes	71.23	70.98	R
Decision Tree	72.94	71.83	R
Random Forest	73	72.5	R
Neural Networks	78.2	77.9	R
SVM	79	79.3	Azure ML
Logistic Regression	81	82.8	Azure ML
Decision Forest	80.5	83	Azure ML
Boosted Decision Tree	85.7	**86.3**	Azure ML

Table 7: Accuracy of different Binary classifiers.

Methods	Accuracy(%)		Built-in
	Test data	Hidden data	
Baseline I	17.98	24.7	R
Baseline II	27.92	79.15	R
Neural Networks	12.4	26	Azure ML
Random Forest	72	60.58	R
Decision Tree	71.94	60.77	R
Naïve Bayes	60.14	62.34	R
Logistic Regression	74.6	**76**	Azure ML
Decision Forest	77	**76**	Azure ML

Table 8: Accuracy of different Multi Class classifiers.

Observation: The above graph represents the ROC plot where the horizontal axis represents the ratio of false positive rate and vertical axis represents the ratio of True positive rate. A higher value on the horizontal axis implies worse performance while a higher value implies a better performance and also makes sure that there is fewer false positive and fewer false negative. URL length, number of dots, suffix, and number of distinct characters are the best features, when removing these features the accuracy reduced significantly from 81% to 76%. The running time for the two-class logistic regression in Azure ML is 5 to 10 minutes.

Multi Class Classifier was built using logistic regression in Azure ML having the **parameters** similar to binary classifier. **Accuracy** of 74.6% was observed for the following parameters, OT :10^{-7}, L1RW :1, L2RW : 1, MemSize : 20.

Observation: Number of distinct characters, suffix, URL length and number of dots are the **best features** for this model. Mean, median of n-gram did not contribute much to the accuracy of the model, the accuracy remained the same even after removing those features.

4. RESULTS AND DISCUSSION

We tried the following machine learning methods on our dataset:

1. Naïve Bayes using R
2. Decision Tree using R
3. Random forest using R
4. Neural Networks using R and Azure ML
5. Boosted Decision Tree using Azure ML
6. Decision Forest using Azure ML
7. Support Vector Machines using Azure ML
8. Logistic Regression using Azure ML

The following are the parameters and accuracy of the best models for the hidden data provided. Classifier: Binary, Method: Boosted Decision Tree, Accuracy on Hidden Test Data: 86.3, Number of trees constructed: 300, Maximum number of leaves per tree: 20, Minimum number of samples per leaf node: 10, Learning rate: 0.2.

Classifier: Multi-class, Best Model: Decision Forest, Accuracy on Hidden Test Data: 76, Re-sampling Method: Bagging, Number of decision trees: 250, Maximum depth of the decision trees: 700, Number of random splits per node: 500, Minimum number of samples per leaf node: 1.

The Baseline II for both binary and multi-class classifier is high. This is because the hidden data is very skewed with 79.15% of non-DGA URLs. The best models are boosted decision tree for binary and decision forest for multi- class classifier. The reason could be that both methods are ensemble methods and as we know ensemble methods usually works better than any other methods and improves the accuracy. The training time of the model increased with 10-fold cross validation due to large amount of data. Hence, 80% of the whole dataset is used to train the models and remaining 20% to test the models.

5. FUTURE WORK

The linguistic features based on dictionary words could be generated. This could add more insights to our analysis. We have experimented ensemble methods using decision trees only. The combination of several methods like KNN, Neural Networks, Decision trees etc., using ensemble could further improve the model. The Neural Networks is build for less number of hidden nodes and other parameters such as the number of hidden layers; the number of learning iterations is kept constant. The model could be improved by varying these parameters.

6. ACKNOWLEDGMENT

First and foremost we would like to thank Prof. Martine de Cock for her invaluable comments. We thank James Andrew Marquadt for providing us with an initial understanding of the problem and providing the dataset. Finally, we thank our peer group members whose comments helped to improve our work.

7. REFERENCES

[1] http://azure.microsoft.com/enus/services/machine-learning/.
[2] http://en.wikipedia.org/wiki/artificial_neural_network.
[3] http://en.wikipedia.org/wiki/support_vector_machine.
[4] https://msdn.microsoft.com/en-us/library/azure/dn905994.aspx.
[5] https://www.python.org/.
[6] http://www.alglib.net/dataanalysis/decisionforest.php.
[7] http://www.r-project.org/.
[8] naive-bayesclassifier.pdf.
[9] M. Antonakakis and P. et al. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX security symposium*, pages 491–506, 2012.
[10] W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews*, 1(1):14–23, 2011.

8. APPENDIX I: MIN-MAX NORMALIZER

Assume that there are n rows with seven variables, A, B, C, D, E, F and G, in the data. We use variable E as an example in the calculations below. The remaining variables in the rows are normalized in the same way. The normalized value of e_i for variable E in the i^{th} row is calculated as:

$$Normalized(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

where, E_{min} = the minimum value for variable E. E_{max} = the maximum value for variable E. If E_{max} is equal to E_{min} then Normalized (e_i) is set to 0.5.

9. APPENDIX II:AZUREML WORKFLOWS

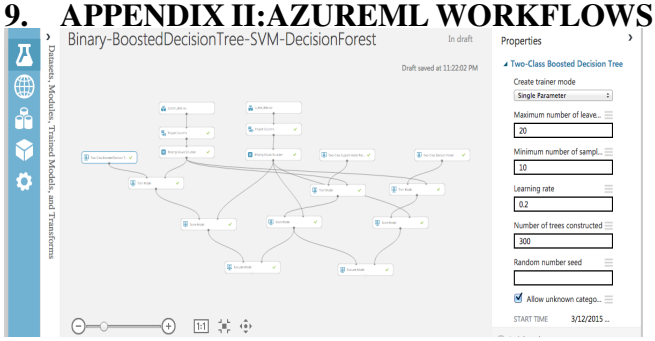

Figure 6: Workflow and parameters for Binary Boosted Decision Tree

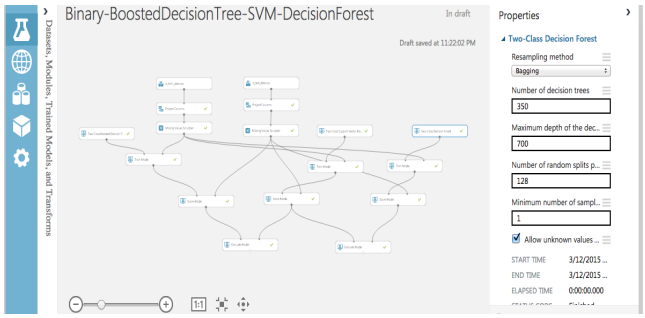

Figure 7: Workflow and parameters for Binary Decision Forest

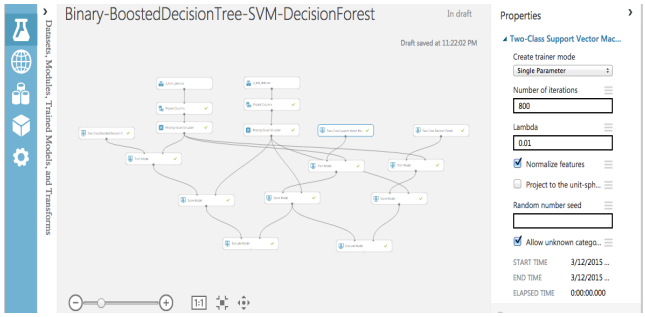

Figure 8: Workflow and parameters for Binary Support Vector Machines

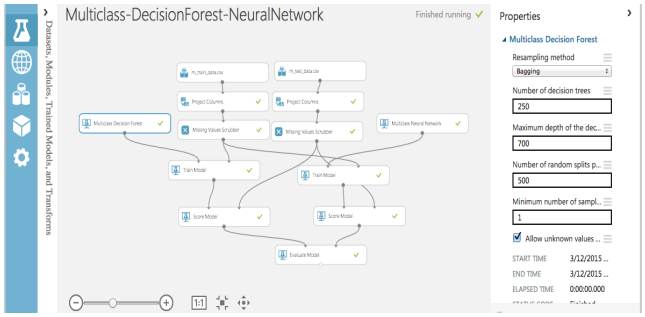

Figure 9: Workflow and parameters for Multi-class Decision Forest

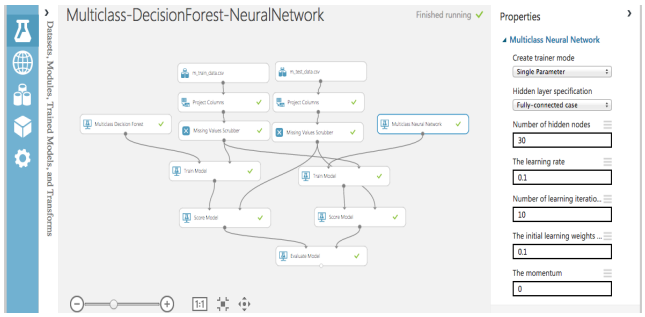

Figure 10: Workflow and parameters for Multi-class Neural Network

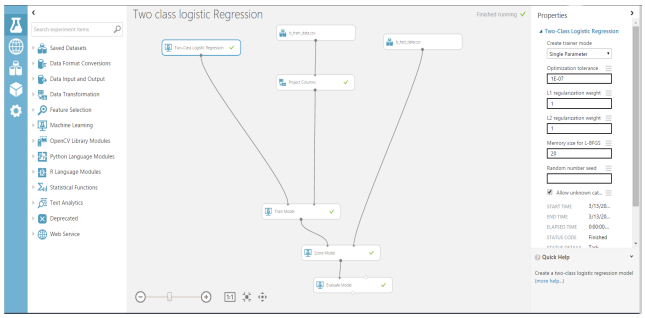

Figure 11: Workflow for Two-class logistic Regression with parameters

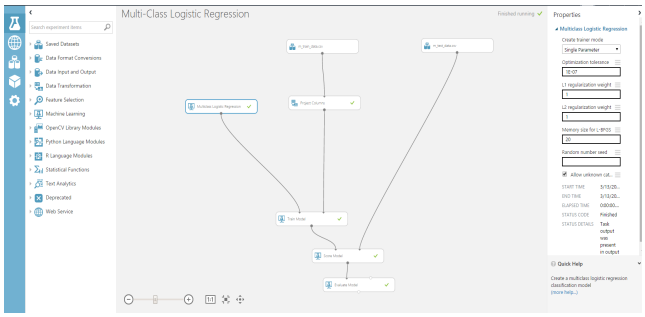

Figure 12: Workflow for Multi-Class logistic Regression with parameters