Microsoft Malware Prediction

Vadiwoo Karuppiah
Department of Computer Science
Syracuse University
New York, USA
vkaruppi@syr.edu

Jose Arevalo
Department of Computer Science
Syracuse University
New York, USA
joareval@syr.edu

Indira Emmadi
Department of Computer Science
Syracuse University
New York, USA
iemmadi@syr.edu

ABSTRACT

Predictive modelling refers to the task of building a model for the target variable as a function of independent variables. There are two types of predictive modeling tasks, classification and regression. In this paper we are focused on the cross-industry process - data mining (CRISP-DM) methodology and have built different predictive models using different supervised classification algorithms. The initial dataset required a lot of preprocessing, as the dataset is highly unbalanced. It is observed that the accuracy of the results depends on data preprocessing, feature selection and the choice of classification algorithm. The results and the performances of the different algorithms are compared and discussed.

Keywords
CRISP-DM; Decision Tree; Logistic Regression; Random Forest; Neural Network

1. INTRODUCTION

Microsoft Malware Prediction is a Kaggle's Research Prediction Competition topic that can be found at on the Kaggle website. The goal of this competition is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. Kaggle has provided the train.csv and test.csv files for training and testing accordingly. Each row in this dataset corresponds to a machine, uniquely identified by a MachineIdentifier. HasDetections is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, we need to predict the value for HasDetections for each machine in test.csv. This paper is organized as follows. Section 2 describes the study objectives. Section 3.1- 3.3 provides the business and data understanding. Section 3.4 contains various models that will be used to perform models. The paper is concluded in the conclusion section with a summary of the findings.

2. STUDY OBJECTIVES

The goal of this study is to use CRISP-DM methodology to predict if a specific machine will be infected by malware and comparing the results with three different supervised classification algorithm, decision tree, random forest and neural networks.

3. CRISP- DM METHODOLOGY

CRISP-DM stands for cross-industry process for data mining. The CRISP-DM methodology provides a structured approach to planning a data mining project. The phase of CRISP-DM is as listed next.

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modelling
5. Evaluation
6. Deployment

3.1 Business Understanding

Microsoft envisioned to build a predictive model that can accurately predict whether a computer would become infected based on the device's configuration. This would help Microsoft to improve the configuration of their system, hence improve the security.

3.2 Data Understanding

The size of the training and testing data is about 9 million and 8 million rows, respectively. This file contains machine configuration information such as the processor type, OS version, Windows Defender engine, whether the firewall, UAC, or SmartScreen is enabled, the amount of RAM and storage capacity. There are 83 features in total, with 53 categorical features and 30 of which are encoded numerically to protect the privacy of the information.

Challenges in Data Preprocessing

3.2.1 Large Dataset

In general R has some problems with large datasets because R reads the entire data set in RAM all at once. This means that an R object lives entirely in memory. R does not have an int64 data type which means it isn't possible to index objects with huge number of rows and columns. In this project one of the team members used a mac computer with 8Gb of RAM and when the entire 10Gb dataset was ingested, the model could not be run because there were memory errors. For this project the fread function was used instead of read.csv because fread function was 500% faster. This speed is attributed to fread memory mapping the file into memory then iterating through the file using pointers. The read.csv function on the other hand reads the file into a buffer via connection so it is much slower.

3.2.2 Data Cleaning

It is very common that datasets contain missing values due to variety of reasons. The train.csv dataset contains huge number of missing values. Figure 3.2.2 shows the dimension of rows in the dataset with missing values.

There are several strategies for dealing with missing data, each of which is appropriate in certain circumstances. These strategies are listed next and each modelling algorithm used the appropriate strategy to build a predictive model.

I. Excluding Missing Values from Analyses
II. Impute missing values with mean, median or mode.
III. Replace missing values with 0.

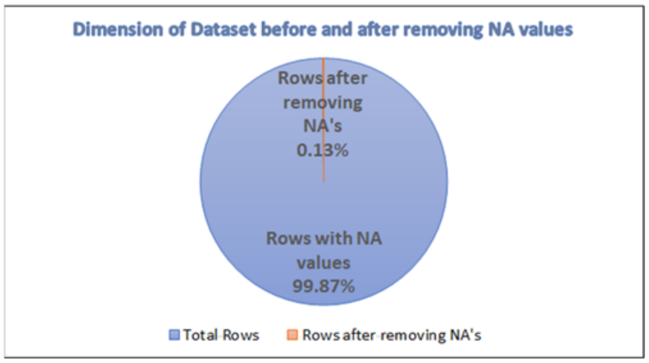


Figure 3.2.2 Percentage after NA removal

3.2.3 Feature Selection

Feature selection is the process of choosing variables that are useful in predicting the target variable. The train.csv dataset contains 82 predictive variables, and it is considered a good practice to identify which features are important when building predictive models. There are various numbers of feature selection approaches that can be implemented in R and we used Boruta and VarImp() function from caret package to identify the important target variables.

1)Boruta

Boruta is a feature ranking and selection algorithm based on random forests algorithm. Figure 3.2.2.1 shows the selected features from Boruta. The values in green are the most important to the algorithm. The attributes closer to 0 are the least important for the accuracy of the dataset.

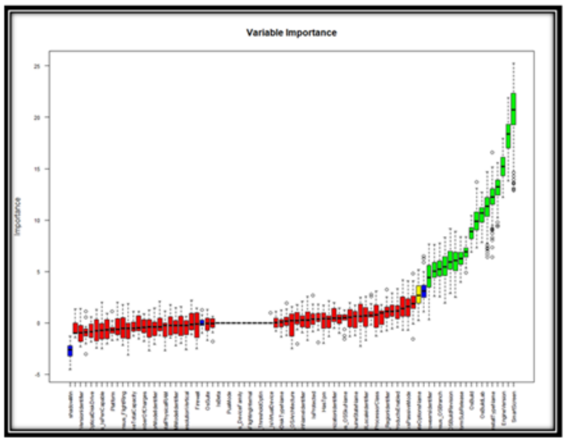
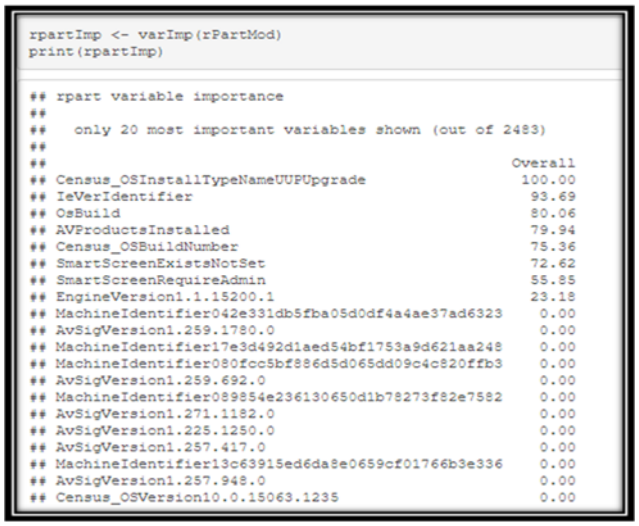


Figure 3.2.2.1: Feature selection from Boruta method.

2) Feature importance from machine learning algorithm.

Another way to look at feature selection is to consider variables most used by various machine learning algorithms the most to be important. We used varImp() function from caret package. At first, we have to train() our model using caret package and then use the varImp() to determine the feature importance.

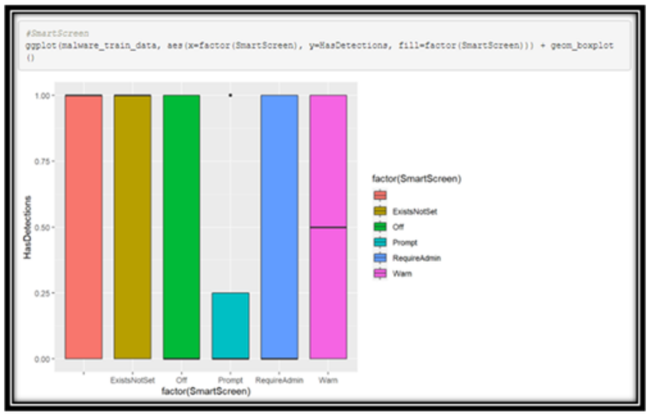


```
rpartImp <- varImp(rPartMod)
print(rpartImp)
```

```
## rpart variable importance
##
##   only 20 most important variables shown (out of 2483)
##
##                                                        Overall
## Census_OSInstallTypeNameUUPUpgrade                     100.00
## IeVerIdentifier                                         93.69
## OsBuild                                                 80.06
## AVProductsInstalled                                     79.94
## Census_OSBuildNumber                                    75.36
## SmartScreenExistsNotSet                                 72.62
## SmartScreenRequireAdmin                                 55.85
## EngineVersion1.1.15200.1                                23.18
## MachineIdentifier042e331db5fba05d0df4a4ae37ad6323        0.00
## AvSigVersion1.259.1780.0                                 0.00
## MachineIdentifier17e3d492d1aed54bf1753a9d621aa248        0.00
## MachineIdentifier080fcc5bf886d5d065dd09c4c820ffb3        0.00
## AvSigVersion1.259.692.0                                  0.00
## MachineIdentifier089854e236130650d1b78273f82e7582        0.00
## AvSigVersion1.271.1182.0                                 0.00
## AvSigVersion1.225.1250.0                                 0.00
## AvSigVersion1.257.417.0                                  0.00
## MachineIdentifier13c63915ed6da8e0659cf01766b3e336        0.00
## AvSigVersion1.257.948.0                                  0.00
## Census_OSVersion10.0.15063.1235                          0.00
```

Figure 3.2.2.2: Feature selection from VarImp() function from caret package.

3.3 Data Visualization



```
#SmartScreen
ggplot(malware_train_data, aes(x=factor(SmartScreen), y=HasDetections, fill=factor(SmartScreen))) + geom_boxplot
()
```

Figure 3.3.1: SmartScreen configurations that causes malware attack.

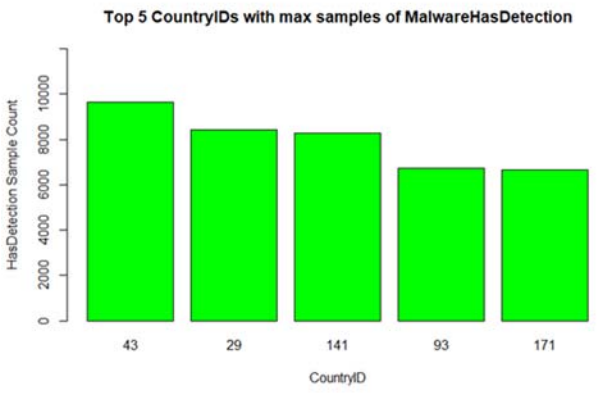


Figure 3.3.2: Top 5 Country IDs seeing the Malware

Figure 3.3.1 and Figure 3.3.2 shows some data visualization that we analyzed from selected attributes. From Figure 3.3.1 we could see that, “ExistsNotSet” attribute from SmartScreen could also cause the malware attack on Microsoft systems.

3.4 Modeling and Evaluation

In this section, we have discussed how predictive models have been built and evaluated. There is no silver bullet in terms of which classification models to be used on datasets. Therefore, choosing appropriate algorithms is an important decision in data mining and machine learning, and it requires knowledge of both the data set and the classification algorithm. We have selected Decision Tree, Logistic Regression, Random Forest and Neural Network modelling for this project.

3.4.1 Decision Tree and Logistic Regression

Decision Tree Classification model is one of the most common data mining models that’s easy to understand. The classification uses Hunt's Algorithm where attributes are split using recursive partitioning approach. It is implemented in RStudio using the rpart package. This algorithm is well suited while dealing with Categorical attributes and binary attributes.

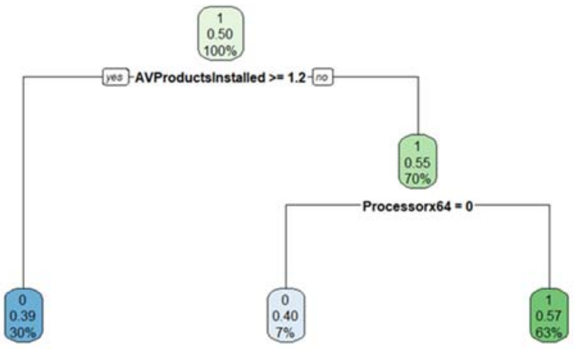
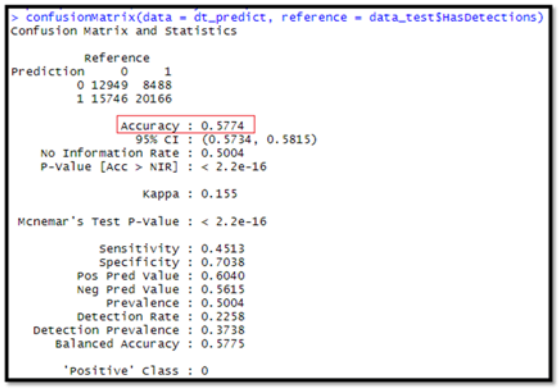


Figure 3.4.1.1: Decision Tree Model

During data preprocessing, no duplicate values were observed. However, there were lots of missing values that are replaced with mean values and dropped the all-zero columns. 17 attributed were selected as a result of feature



```
> confusionMatrix(data = dt_predict, reference = data_test$HasDetections)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 12949  8488
         1 15746 20166

               Accuracy : 0.5774
                 95% CI : (0.5734, 0.5815)
    No Information Rate : 0.5004
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.155

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4513
            Specificity : 0.7038
         Pos Pred Value : 0.6040
         Neg Pred Value : 0.5615
             Prevalence : 0.5004
         Detection Rate : 0.2258
   Detection Prevalence : 0.3738
      Balanced Accuracy : 0.5775

       'Positive' Class : 0
```

Figure 3.4.1.2: 57% Accuracy with Decision Tree Classification

The accuracy of the model is obtained as 56%. Pruning is performed with cp = 0.2, to reduce the size of the Tree and tuning is performed on the model with minsplit = 4, minbucket = round (5 / 3), maxdepth = 3, cp = 0. The accuracy is very slightly improved to 57%.

Using rattle package in RStudio, Logistic Regression is applied on the same dataset.

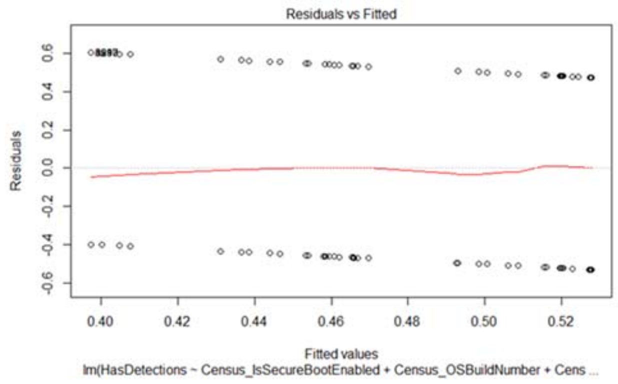
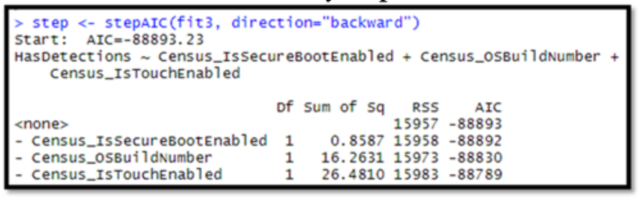


Figure 3.4.1.3: Logistics Regression Model

Logistic regression in its basic form uses a logistic function to model a binary dependent variable.



```
> step <- stepAIC(fit3, direction="backward")
Start:  AIC=-88893.23
HasDetections ~ Census_IsSecureBootEnabled + Census_OSBuildNumber + 
    Census_IsTouchEnabled

                            Df Sum of Sq   RSS    AIC
<none>                                   15957 -88893
- Census_IsSecureBootEnabled  1    0.8587 15958 -88892
- Census_OSBuildNumber        1   16.2631 15973 -88830
- Census_IsTouchEnabled       1   26.4810 15983 -88789
```

Figure 3.4.1.3: Tuned Model: Fitting Parameters of highly correlated attributes

Figure 3.4.1.3 shows that the top 3 highly correlated attributes are taken for modeling.

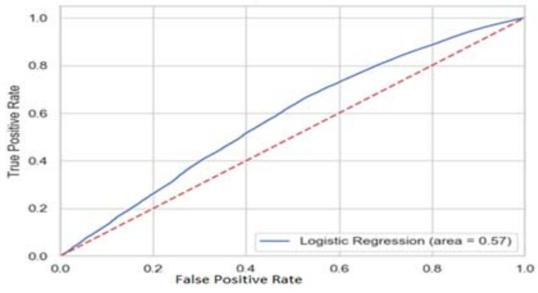


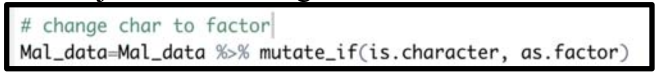
Figure 3.4.1.4 shows that the accuracy hasn't changed much with logistic regression.

3.4.2 Random Forest

Random Forest is a supervised algorithm that consist of multiple decision trees. The number of trees and the accuracy in Random Forest algorithm have a direct relationship where the more trees there are in the model the more accurate it will be. A big difference between Random Forest and Decision Tree algorithm is the process of finding root nodes and splitting of the attribute nodes is done randomly. Some major advantages of Random Forest are that it can't be used for both classification and regression. The classifiers in it can handle missing values and can be modeled for categorical values. Overfitting seems to be a double-edged sword for this algorithm because overfitting can lead to worse results. However, if there are enough trees in the model the classifier cannot overfit the model. The RandomForest algorithm from CRAN implements Breiman's random forest which can also be used in unsupervised mode for assessing proximities among data points.

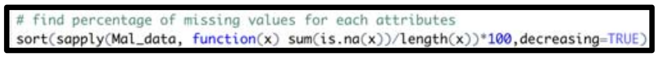
Data preprocessing for Random Forest

The data was a mix of numeric and char objects, in order to use random forest from CRAN all the char objects were changed to factors.



```
# change char to factor
Mal_data=Mal_data %>% mutate_if(is.character, as.factor)
```

Once all these were transformed to factors, the attributes with over 53 levels were removed as the CRAN library for random forest cannot handle over 53 levels. The next step in preprocessing was to look for the percentage of missing values in the attributes.

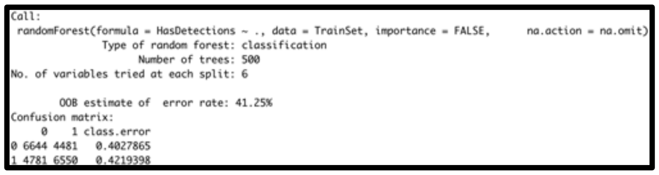


```
# find percentage of missing values for each attributes
sort(sapply(Mal_data, function(x) sum(is.na(x))/length(x))*100,decreasing=TRUE)
```

Any attributes with over 60 percent of its values missing were removed from the dataset. Next step in preprocessing was to plot different attributes against the HasDetection attribute. These plots led to a few more attributes being trimmed from dataset.

Training Random Forest with base model

The dataset was split 70% for training and 30% for validation. Once the data was split up, the below code was used to get the base model. The model error was at 41.25% with an accuracy of 57.5% per the confusion matrix of the validation set. The accuracy isn't that great but not bad for our base mode.



```
Call:
 randomForest(formula = HasDetections ~ ., data = TrainSet, importance = FALSE,      na.action = na.omit)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 41.25%
Confusion matrix:
     0    1 class.error
0 6644 4481   0.4027865
1 4781 6550   0.4219398
```

After seeing these results, I decided to use the importance and variance functions of random forest to see if there are more attributes that can be trimmed before tuning. The below figures show the importance variance plot for the model. A few more attributes were trimmed due to not being important according to the plot.

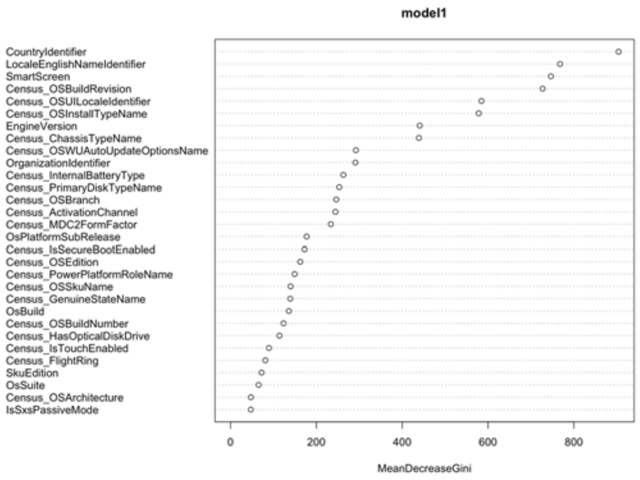


Figure 3.4.2.1: Importance variance plot.

Visualize Results

The below plot shows the random forest trees and how the error decreases. This dataset error seems to plateau after about 400 trees. With tuning the model prediction should improve the model.

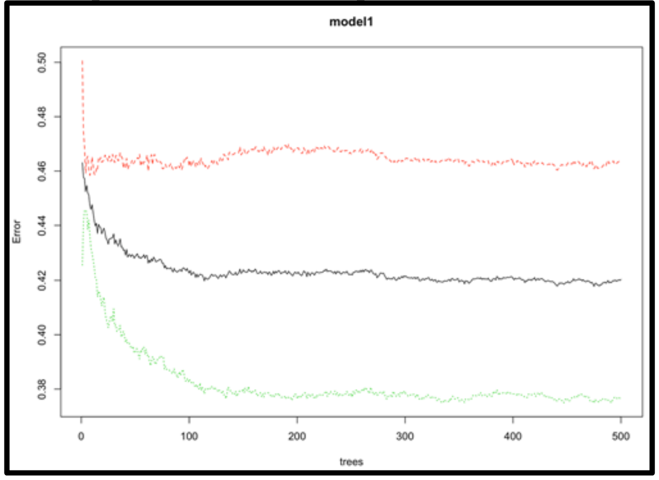


Figure 3.4.2.2: Random Forest plot.

Tuning parameters

In randomForest the parameters that seem most important to tuning are the ntree and mtry. The ntree parameter is the number of trees to grow and the mtry parameter is the number of random variables sampled as candidates at each split. The below table shows how tuning changed the accuracy of the models.

mtry	#Tree	Accuracy(%)
2	300	59.05
3	300	58.36
7	500	57.3
9	500	57.26
5	700	57.84
3	1500	58.4
2	1500	59.1
2	2500	59.2

Table 3.4.2.1 Tuning parameters

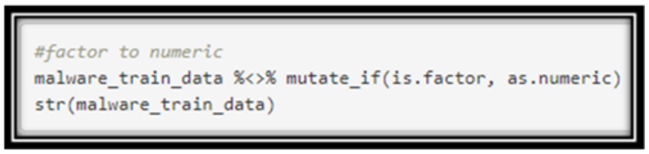
From the table we can see that the nominal split for this dataset is 2 and as the number of trees increases so does the accuracy up to a certain point. For this data set there is obvious challenges and on the site the highest submission have accuracy numbers in the mid 60s. The highest accuracy achieved for random forest was 59.2%

3.4.3 Neural Network

Neural Network is loosely inspired by biological neural network that consists of neurons and synapses. Neural Network consists of input layers and output layers as well as a hidden layer consisting of units that transform the input into something that the output layer can use. We used neuralnet package to train our model for this project. Theoretically neuralnet can handle an arbitrary number of predictor variables and response variables as well as of hidden layers and hidden neurons even though the computational costs can increase exponentially with higher order of complexity. In addition to that, the neuralnet package also provides custom-choice of activation and error function which could improve the results in prediction.

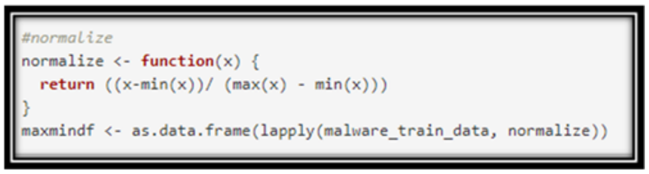
Data preprocessing for neural network

The input data has to be handled effectively and efficiently in developing neural networks. Some important data preprocessing needed for neural networks are data transformation and normalization. The data transformation involved with this dataset was converting categorical into numerical values as shown next.



```
#factor to numeric
malware_train_data %<>% mutate_if(is.factor, as.numeric)
str(malware_train_data)
```

We also performed data normalization with this dataset to ensure that the statistical distribution of values for each net input and output is roughly uniform. In addition, the values should be scaled to match the range of the input neurons.



```
#normalize
normalize <- function(x) {
  return ((x-min(x))/ (max(x) - min(x)))
}
maxmindf <- as.data.frame(lapply(malware_train_data, normalize))
```

Training of neural networks

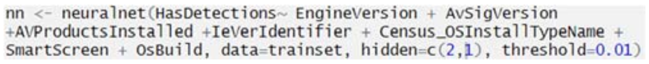
```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
  stepmax = 1e+05, rep = 1, startweights = NULL,
  learningrate.limit = NULL, learningrate.factor = list(minus = 0.5,
  plus = 1.2), learningrate = NULL, lifesign = "none",
  lifesign.step = 1000, algorithm = "rprop+", err.fct = "sse",
  act.fct = "logistic", linear.output = TRUE, exclude = NULL,
  constant.weights = NULL, likelihood = FALSE)
```

- *formula*, a symbolic description of the model to be fitted.
- *data*, a data frame containing the variables specified in formula.
- *hidden*, a vector specifying the number of hidden layers and hidden neurons in each layer. For example, the vector (3,2,1)

induces a neural network with three hidden layers, the first one with three, the second one with two and the third one with one hidden neuron. Default: 1.

- *threshold,* an integer specifying the threshold for the partial derivatives of the error function as stopping criteria. Default: 0.01
- *linear. Output,* logical. If act.fct should not be applied to the output neurons, linear.output has to be TRUE. Default: TRUE

We used the predictor variables from the feature selection method to feed into our neural network as shown below.



```
nn <- neuralnet(HasDetections~ EngineVersion + AvSigVersion
+AVProductsInstalled +IeVerIdentifier + Census_OSInstallTypeName +
SmartScreen + OsBuild, data=trainset, hidden=c(2,1), threshold=0.01)
```

Visualizing the results

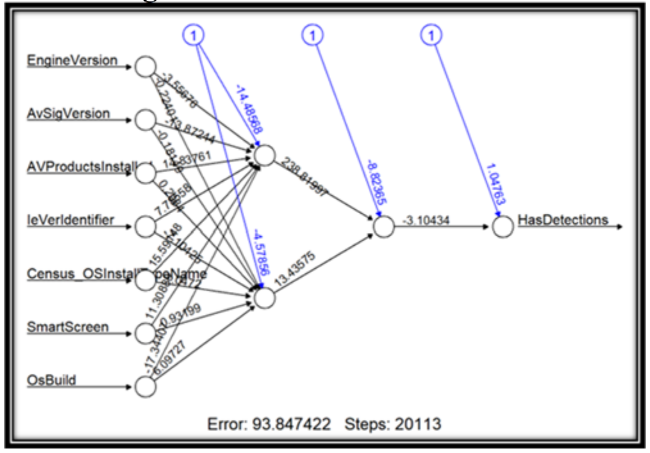
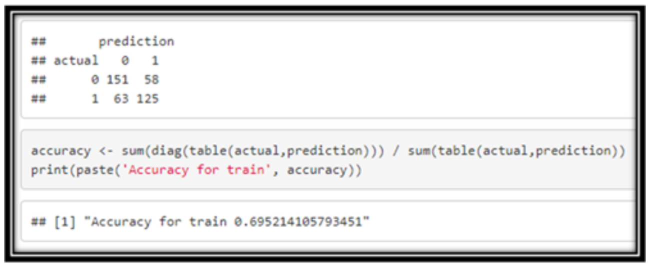


Figure 3.4.3.1 : plot of neural network model



```
##        prediction
## actual   0   1
##      0 151  58
##      1  63 125
```

```
accuracy <- sum(diag(table(actual,prediction))) / sum(table(actual,prediction))
print(paste('Accuracy for train', accuracy))
```

```
## [1] "Accuracy for train 0.695214105793451"
```

Figure 3.4.3.2: confusion matrix for neural network model

Accuracy

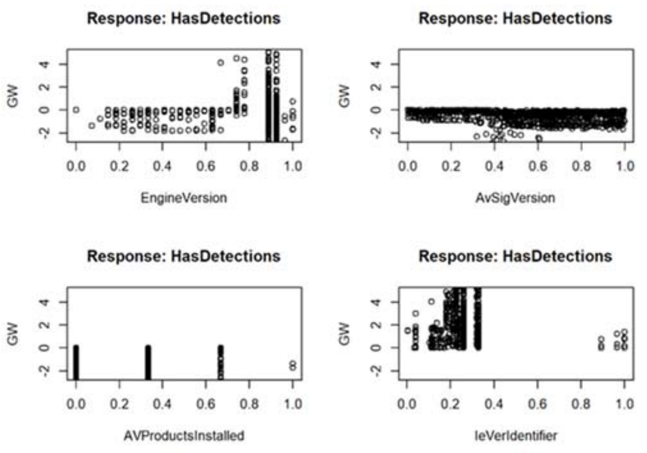
The neural network model could achieve accuracy of 69.5% as shown from Figure 3.4.3.2. The same model was tested on multiple samples from the dataset and with the entire dataset, train.csv.

Data Sample	**Total Observations**	**Accuracy(%)**
TrainSample1.csv	1 000 000	69.5
TrainSample3.csv	1 000 000	68.2
TrainSample5.csv	1 000 000	65.0
TrainSample7.csv	1 000 000	69.1
Train.csv (full dataset)	8 921 483	68.7

Figure 3.4.3.2: Accuracy for different subsets of train.csv file

Generalized weights

Generalized weights for all observations can be visited in "generalized.weights" element in the nn class object returned by neuralnet() function. Graphical visualization of generalized weights is another way to examine the relative contribution of each target variable as shown in Figure 3.4.3.3.



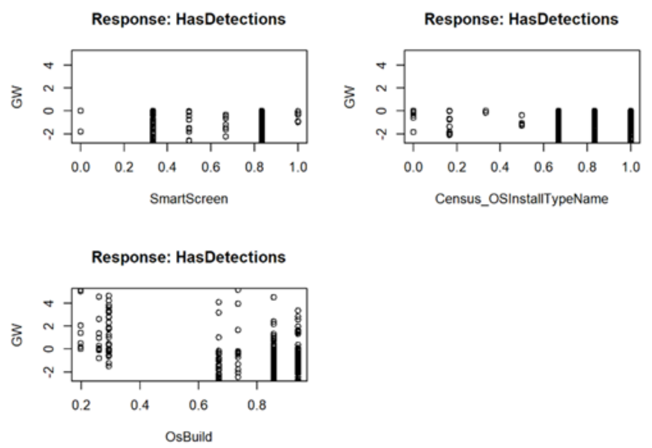


Figure 3.4.3.3: generalized weights for all the target variables.

Hyper Parameter Tuning for Neural Network

Hyper-parameter is a configurable value which is set before the learning process begins. These hyper-parameter values dictate the behavior of the training algorithm and how it learns the parameters from the data. There are various parameters that can be tuned for neural network as shown below:

- Number of hidden layers
- Number of nodes in each layer
- Loss function
- Activation function
- Optimization method (e.g. gradient descent)
- Epoch Mini-batch or individual update weights
- Regularization: L1/L2 form, dropout rate

We have selected to tune the hidden layers and activation function of our neural network model.

1) Tuning hidden layers and number of nodes in each layer.

Theoretically, more hidden layers and number of neurons per layer can improve the performance of neural network. However, our model did not improve with more hidden layers and neurons as shown in Figure 3.4.3.4. It was also noticed for some tests, the weight optimization fails, and the neural network failed to converge.

Hidden Layer	Nodes Per layer	Accuracy (%)
3	2	65.8%
4	1	Failed to converge
5	1	Failed to converge

Figure 3.4.3.4: Tuning Model with hidden layers and neurons.

2)Tuning activation function

The activation function is considered as a mathematical gate in between the input feeding from the current neuron and its output going to the next layer. It is a simple step function that turns the neuron output on and off, depending on a rule or threshold. Some common activation functions are Sigmoid, TanH (Hyperbolic Tangent) and ReLU (Rectified Linear Unit). However, Sigmoid and Tanh suffered from vanishing gradients problems. Rectified Linear Unit (ReLU) is the most widely used activation function as it solves the problem of vanishing gradients.
We have tuned our model using ReLU activation function and the plot of this model is as shown in Figure 3.4.3.1.

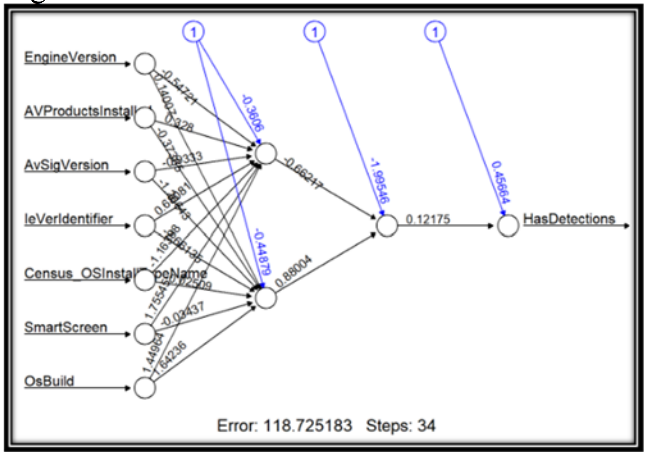


Figure 3.4.3.5: neural network model with activation function.

With activation function, the total steps it took for convergence has reduced tremendously from 20113(Figure 3.4.3.1) steps to 34 steps

(Figure3.4.3.5), however the accuracy has dropped to 51.6%.

4. CONCLUSION

After implementing the CRISP-DM methodology on the dataset with various classification models, it is observed that the accuracy of the model is depending on data-preprocessing, feature selection, classification algorithm and the tuning parameters of the model.

Classification Model	**Total Observations Sample Count**	**Accuracy (%)**
Decision Tree	20,300	56%
Logistic Regression	44,800	57%
Random Forest	1,000,000	59%
ANN	1,000,000	69%

Upon comparing the results of each model, it is seen that the Artificial Neural Networks classification model gave the best accuracy.

5. ACKNOWLEDGMENTS

We would like to thank Professor. Lin for giving us the opportunity to work on this project. It has given us great exposure to data mining topics and techniques.

6. REFERENCES

[1] Zhongheng Zhang, 2016, Neural networks: further insights into error function, generalized weight and others.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5009026/

[2] Frauke Gunther and Stephan Fritish, neuralnet : Training of Neural Networks.
https://journal.r-project.org/archive/2010/RJ-2010-006/RJ-2010-006.pdf

[3] Rohith Gandhi, Improving the Performance of neural network,.
https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d

[4] Microsoft Malware Prediction, https://www.kaggle.com/c/microsoft-malware-prediction

[5] Liaw and M. Wiener. Classification and regression by randomForest. R News, 2(3):18–22, 2002.

[6] Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. In Advances in Neural Information Processing Systems, pages 431–439, 2013.

[7] Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Chapman & Hall, New York, 1984.