Clinton Page

Defence Industry Cyber Challenge

ThreatMetrix Account Takeover Challenge

Introduction

I decided to tackle this part of the challenge by training a feed forward reverse propagation Deep Neural Network (DNN) to classify the malicious transactions. The dataset provided consists of ~1.1 million entries which should be sufficient to train a DNN. I choose to use a DNN because it is an area I am familiar with and these algorithms have been shown to perform well in classification problems.

At its core, a DNN is made of thousands of simple processing units called nodes. A good analogy is to relate to the human brain. Each node is like a neuron in the brain. It receives input signals from other nodes and the applies an activation function which determines if the node should "fire" or not. Now, a single node by itself is not very useful however when you join thousands, millions or billions of nodes together into a network, emergent behaviours start to appear. This network of many nodes is what is called a Deep Neural Network (DNN). It's important to understand that not all nodes in the DNN are identical. The input connections to each node are assigned weights. During the training phase, the input weights in the model are adjusted as the model tries to determine what inputs correspond to the correct output classification. This is fundamentally how a DNN "learns".

Cleaning up the dataset

It is important to remove redundant features from the dataset to ensure the DNN training algorithm converges.

Below is a summary of the features I removed/modified and why:
Datetime – modified to be the time span from *fuzzy device_first_*seen. Typically, you don't want your malicious classifier to classify a transaction as malicious based on a specific date. Instead of discarding this value we can replace it with the timespan between when the timestamp and the *fuzzy device_first_*seen time.
device_first_seen – modified to be the time span from *fuzzy_device_first_seen*. Following the same logic as DateTime the device_first_seen field was replace by the length of time from the *fuzzy_device_first_seen* time.
true_ip – removed. IPv4 addresses are typically dynamic and constantly change over time. Making a classifier based on IP could improve classification but will not generalise well (i.e. the classifier will perform poorly for future transactions).

Distributing the Dataset

The dataset is skew towards genuine entries where only 0.67% of entries are labelled as malicious. It is important this is taken into account otherwise the training algorithm will be heavily bias towards genuine entries.

To address this issue all malicious entries were duplicated 5 times (leading to a total of 39K malicious test cases) and only every 1 in 25 genuine samples were sampled (resulting in 47K genuine test cases). If I had more time I would make sure the genuine samples selected represent a wide range of entries however to keep it simple I just shuffled the genuine entries and selected 1 in every 25th entry.

The dataset was then split into two files with 75% of records allocated for training and 25% for testing. The training dataset is used for training the model and the testing dataset is used the evaluate the performance of the classifier. All results mentioned in the results section are derived from the 25% testing dataset.

Calculating the Classification Model

The number of nodes and layers is the model was then calculated based on the dataset size available. As a general principle, the number of variables to train (known as model complexity) should be less than half the dataset. The reasoning behind this is every variable in the model should be trained to be between an upper and lower boundary. Therefore, two test cases are needed for each training variable. The model complexity of a DNN is expressed in equation 1 below where N is the total number of layers in the DNN and L_i is the number of nodes at layer i:

$$Model\ Complexity = \sum_{i=0}^{i=N} L_{2i} \times L_{2i+1} \qquad \textit{Eq. 1}$$

Therefore if L=[21, 100, 100, 2] (i.e. first layer has 21 nodes, second layer has 100, etc) then the model complexity is 12.3K which satisfies the general principle stated above (there are 60K entries in the training dataset).

Table 1, summaries the parameters chosen for the DNN classifier.

Model parameter	Description	Configuration
Data representation	How the input and output data is encoded.	1-hot-encoding was used for inputs that have no intrinsic order (e.g. language, OS, etc). Fields that have a clear order (e.g. time values, amounts, etc) were normalised such that all values in the range were between 0.0 – 1.0. This encourages convergent behaviours when training the model.
Numb. of nodes in each layer	The number of nodes in the model. As the number of nodes increases so does the model complexity. Too many nodes can result in over-fitting while not enough can result in under-fitting.	21 nodes in the input layer (fixed from the input encoding), 100 nodes in the first hidden layer, 100 in the second hidden layer and 2 in the output layer (1 for malicious and the other for genuine).
Training algorithm	Also known as the optimization algorithm. This is parameter controls how the weights are adjusted after each iteration.	I choose the Adam optimization algorithm because of its tolerance to ill-conditions and computation efficiency.
Activation functions	Responsible for detecting non-linear patterns in the input data.	Rectified Linear Unit (ReLU) and Softmax at the output layer.

Table 1 *–The model parameters used in constructing the DNN.*

The Python library Tensorflow was used to construct the DNN. Please refer to: [Defence Industry Cyber Challenge Repository]() for my implementation.

Results

The overall **classification accuracy** after training my DNN wa**s 83.77%**. This means for every 100 transactions on average 84 transactions will be correctly classified. The Confusion matrix (Table 2) and Precision and Recall graph (Figure 1) are:-

	Predicted genuine	Predicted malicious
Actual genuine	**87%**	**13%**
Actual malicious	**21%**	**79%**

Table 2 *- The confusion matrix of the classifier. Columns show the model's prediction with rows showing ground truth output label. Therefore 87% of genuine entries were correctly identified and the remaining 13% were classified incorrectly as malicious.*

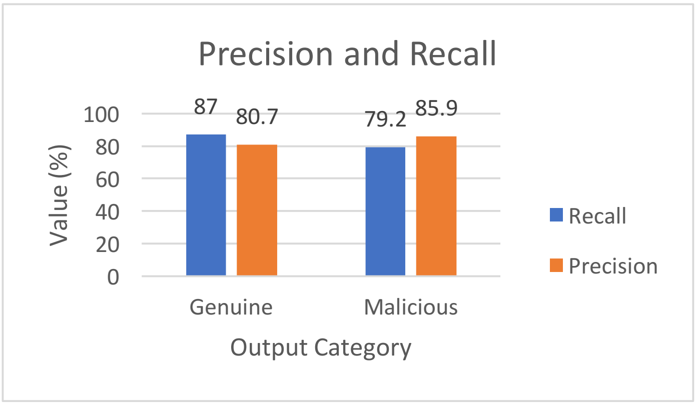

Figure 1 *– The precision and recall results for both genuine and malicious categories. the malicious precision (85.9%) was slightly higher than its recall. This implies that when the classifier's prediction was malicious then the probability of this being correct is 85.9%.*

Overall the results collected where positive indicated that DNN is a viable solution for the classification problem. Future work could include exploring using an auto-encoder with the discarded entries and further analysis on how the input weights effected the output classification.

Thank you for hosting the Defence Industry Cyber Challenge which gave me the opportunity to solve interesting problems related to cyber security.