CS 185C HMM Midterm 2

Preprocessing Step

The original data consisted of three main families of malware: winwebsec, zbot, and zeroaccess. Each family has thousands of files and each file contains tens of thousands of lines. In order to train the model, we split each family according to an 80-20 split. Eighty percent of the files and data will be used to train the model, and the remaining twenty percent is used to test the model. This was done through a python library called 'splitfolders'. It randomly selects files in each family to be split, which is controlled through a seed and outputs the families into training and testing sets.

The next step is to generate a dictionary for each family in the order of occurrence for each line type (i.e 'mov, 'push, etc). For each family, we read their training dataset and iterate through to count the number of occurrences. We then sort the dictionary in descending order to get the highest count to lowest count. Now, there exists a dictionary that can be indexed and used for our model.

Tuning

There were three main hyperparameters used for tuning the model: number of states, number of observation symbols, and stop conditions.

The number of states in our model are not determined. In order to find the most optimal number of hidden states, we train the model using our algorithm and then compute the log likelihood according to the model. We repeat these steps to find the maximum log likelihood which would then be the best number of states.

The number of observation symbols are the indexes we use to redefine our dictionary. From our preprocessing step, we obtained a dictionary that is ordered from most to least used function names in each family. By converting the names into numbers, we are able to use them to train our model. What needs to be determined is what height of index do we use. This is because the first few functions may occur hundreds of thousands of times, whereas the last few only occur tens of times. We must merge the lower end of functions together into a single index to be comparable to the top functions.

The stop conditions are determined by two ways. The first is through the number of iterations. Our algorithm can run for a certain number of runs, and then stop when reaching that number.

The finalized training model is then used for testing. The second stop condition is when the increase of $P(\sigma|\lambda)$is too small. We can determine what the value of too small is in order to adjust our stopping condition.

Generally, increasing the number of states was not improving the overall accuracy of the model to a significant degree. Furthermore, increasing the number of states drastically increased the training time of the model. Since changing the number of states around was not cost effective, the majority of testing and training was done with N=2.

Our stopping criteria was done such that if the log probability was not increasing in the next iteration, then we would stop. Otherwise, the total number of iterations done for training is set at 100.

The most variable factor in our model training was done through changing the indexes. Our initial index was set at 15, meaning that the most frequent function is 0 and the remaining grouping of functions would be 15. This results in a total of 16 unique observation symbols. As we continue to try to produce a better model, we increase the index count incrementally. In total, the training was done with indexes from 15-19. This results in a total of five models per family.

Experimentation

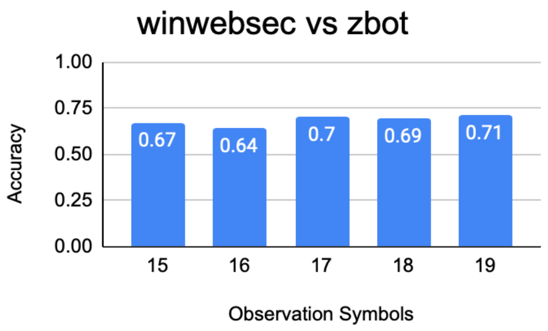
The process of experimentation first begins with model training. The model is trained in respect to the observed training sequence, an initial state transition probability matrix (A), an initial observation probability matrix (B), and an initial state distribution (π). In each test that is performed, the model generates random A,B, andπvalues. The values are stochastic, but not uniform since perfectly uniform values result in convergence to a local maximum.

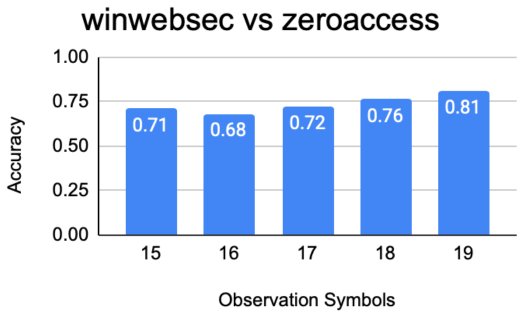
Each test is done through creating models based on training on one dataset and compared against another dataset. After training, we can use the forward algorithm to produce a log likelihood value which we then normalize and then calculate the probability. The reason behind normalization is due to the fact that a smaller log likelihood is length dependent. Therefore winwebsec with over four thousand files cannot be directly compared to zeroaccess with a thousand files without normalized scores first. The probabilities are then generated using scikit's roc_auc_score library.

There are three tests that are being performed. The first is training a model on a winwebsec dictionary and then comparing against zbot and zeroaccess. The second is training on a zbot dictionary and then comparing against winwebsec and zeroaccess. The last is training on a zeroaccess dictionary and then comparing against winwebsec and zbot.

Test 1 WinWebSec

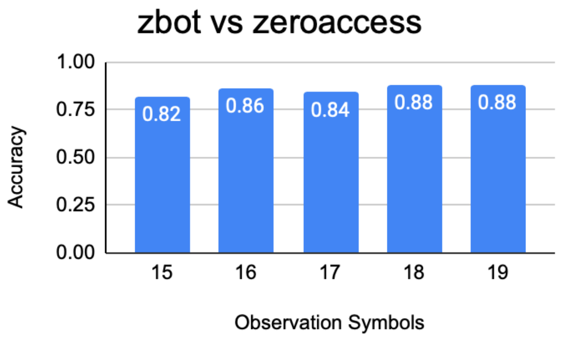
The first test done is through winwebsec's dictionary. The graphs below depict the probabilities that were generated from the other two families in comparison to winwebsec. What we can see is that given a model that is trained from winwebsec, the model is more accurately able to determine a sequence from zeroaccess than it is able to from zbot.

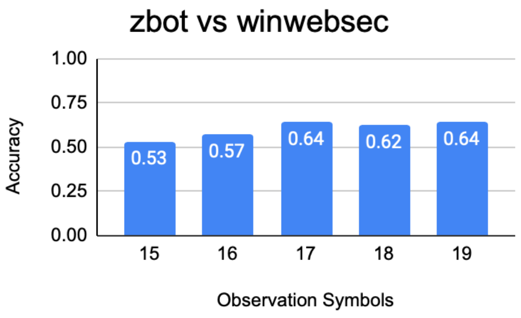




Test 2 ZBot

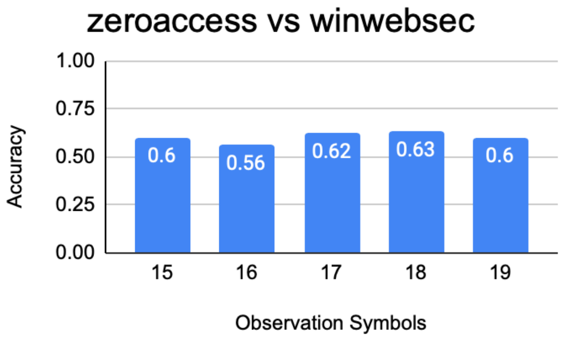
The second test done is through zbot's dictionary. The graph below depicts the probabilities and show a clear differentiation in the model's ability to predict sequence from other families. In comparison on zeroaccess and winwebsec, a model trained on zbot's data will have a higher probability of determining zeroaccess than it will winwebsec.

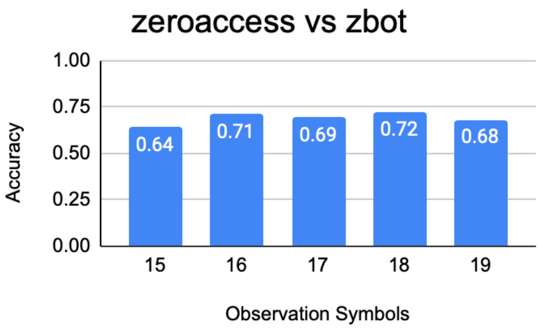




Test 3 ZeroAccess

The third test is done through zeroaccess' dictionary. The graph depicts probabilities are similar, but with zbot having slightly higher accuracy in comparison to winwebsec.





Something to consider is the fact the length of training data is different for each family so there could be some variance as a result. For example, winwebsec's total files exceed four thousand. Taking 80% of that for training is still above both zeroaccess' and zbot's total files. This variance could affect the result when we inserted the testing data for each family into the model.

Conclusion

In our experiments we have found probabilities of malware detection in comparison from one family to another. The data comes from three main malware families: winwebsec, zbot, and zeroaccess. Each family is split into training and testing data which are then indexed into dictionaries and converted into their respective observation symbols which are used in HMM training. The resulting data collected shows the best comparison between zbot and zeroaccess when training on zbot's data. For future experiments, there are a few things that can be done. The preprocessing step in this experiment was done through a library that randomly split files into training and testing. It could be more viable if we performed some form of n-folds cross validation onto the data in order to encapsulate the data better. In terms of algorithms, there can be some application of ensemble learning in order to improve training accuracy. Lastly, there can be implementation of different techniques in comparison to HMM's to show some comparison between them.