CS185C:
Final Project
Malware Classification

Jordan Conragan, Brett Dispoto

May 10, 2020

Contents

Part I
Preprocessing

1 Preprocessing the Dataset

Most of the machine learning techniques used in this report use variations of the same preprocessing steps. Here are the preprocessing steps taken for the methods described in this report. The preprocessing was done in python3, and the relevant files can be found in the `preprocessing` directory of our submission.

1. Download the dataset.

2. Split the dataset into directories based upon their family label. (Already completed by the dataset provider.)

3. For each malware family, the following steps were then taken:

   (a) Read through all of the files, count the occurance of each uniqe opcode across all files.

   (b) Take the n (turning parameter) most common Opcodes, and convert them to an ASCII symbol for observation sybmols for our HMMs. The Opcodes which are not within the n most common will be converted to an "other" symbol. This will reduce noise in our model.

   (c) Once each opcode is assigned a symbol, we again read through the files and convert the opcodes to symbols.

       i. If bagging is being used, make copies of **each** converted malware file, which will later be split up accordingly during training.

       ii. Otherwise, if boosting or stacking is being used, we can simply dump the converted opcodes (symbols) for the entire family into one huge file. This file will be our observation sequence.

Part II
Experiments

2 Bagging

Bagging is performed using the files `Bagging.java` and `HiddenMarkovModel.java`. The procedues and results are described in the secions below.

2.1 Bagging Procedure

The following steps were performed in order to use bagging as an ensemble method:

1. Split assembly instructions into their appropape family, and translate the instrutins to HMM symbols,

2. Within these family folders, use a shell script (included, `makeTesting.sh`) in order to split 10 percent of the samples into a test set.

3. Train x HMMs on each family,

4. Once each HMM for a given family is trained, score the observation sequence which just used to train the model.

5. Write down this score.

6. Once all HMMs of a given family are trained, use whichever ensembler aggegrate function to take all the generated scores and aggregate them into one score.

7. Once training is complete for all families, for each family we now have this "aggregate score" written down in a log in the directory where the training samples are located.

8. Finally, to test all of these trained HMMs, we go into all of the test sets, score the test sample using the **each of the three "ensemblers"** (each made up of x HMMs.). Once we have the score from each ensembler, we can then go back to the logs where the original score was written down for the training samples.

9. Now, our sample has one score for each ensembler, denoted as $S_{1..\texttt{num\_ensembler}}(x_{\text{test}})$

10. We classify this sample as whichever has the minimum of $abs(S_{family}(x_{\text{test}}) - score_{\text{family}})$, that is, we classify as whichever family's "original score" score is closest to the score for the sample given by *that family's* "ensembled" HMMs.

2.2 Bagging Results

2.2.1 Initial Bagging Results

Initial bagging results are given in the below table:

Key	Value
Ensembler Aggregate Function	MAX
HMM's per ensembler	10
N (HMM parameter – # states)	2
Random initialization seed (HMM)	0
Test set size	10% of training set
Test Accuracy	0.9974326059050064

As is shown in the table above, the first bagging tests went well. Screenshots of the information described here can be found in figure [1] in the appendix.

1. Whenever a sample was scored **using an HMM which was NOT trained on the family which the sample belongs to, the score is returned as NaN.**

   - For example, if a `winwebsec` sample was scored using an HMM which was trained on samples form the `zbot` family, then all the `zbot` HMMs will give `winwebsec` samples a score of NaN. *This was somewhat unexpected, because our HiddenMarkovModel implementation takes special caution (using logrythims) to avoid underflow. Further, we know our HMM is valid because of extensive testing using Mark Stamp's paper "A Revealing Introduction to Hidden Markov Models".* Our intuition tells us that this is okay, because the samples are still being given a valid score when the correct family's HMM's is used. This leads us to belive that the other probabilities are so, so low, that they are are "log-underflow-trick" resistant.

   - To attempt to remedy this, we tried changing training methodology:

     (a) **Increase number of HMMs** — Currently, durring training, when an HMM is trained on an observation sequence, that observation sequence is much, much larger than the actual samples which are being scored. To remedy this, we can split our "ensemblers" into more "bags" such that they're made up of more HMMs. In turn, each HMM will be trained on a closer amount of data it will be tested on. Despite our rationale.... results from splitting the ensemblers into 30 or 100 HMMs each rather than 10 yielded the *exact same results..*

A   Selected Screenshots

Figure 1: Initial results from Bagging.