Practical 2: Malware Classification

Ethan Alley, Grigory Khimulya, Walter Martin
alley@college.harvard.edu, khimulya@college.harvard.edu, wmartin@college.harvard.edu

March 10, 2017

1 Technical Approach

We explored the problem of malware classification from several different angles:

- Feature engineering
- Running through a variety of models quickly to test which show the most immediate promise
- More focused tuning of hyperparameters for neural nets

Feature engineering process (Grigory and Ethan)

With several different feature sets to test on, we tested out classification with a random forest, a support vector machine, a standard neural net (sklearn's MLPClassifier), and an LSTM net.

Tuning standard neural net parameters was an accessory step, more with the intention of learning about overfitting rather than improving performance. The parameters tuned were:

- Number of hidden layers (1 or 2)
- Size of hidden layers (between 800 and 50)
- Activation function (tanh, relu, or logistic)
- Maximum number of optimization iterations

2 Results

We had a wide variety of performance across the methods and feature sets we tested. For the models that we submitted to Kaggle, our best result was approximately .79 accuracy on the private tests, produced by a random forest classifier.

Model	Acc.
RF, $n = 100$, ON TFIDF	0.791
DEEP NET ON TFIDF	0.785
SVM ON 4G BOW	0.772
SVM ON GBOW	0.735
SVM ON TOPIC MODELED FV	0.342
PASSIVE AGGRESSIVE ON GBOW	0.150

Table 1: Model accuracy on private tests.

In this table, GBOW is "garbage bag of words," FV is "feature vector," 4G is "4-gram," and TFIDF is [Ethan help!]

We also did some tuning of neural net parameters. These cross-validation scores were generated using sklearn's "cross val score" function; we took the average of 5-fold CV results. We were using the "tanh" activation unless otherwise specified. The numbers associated with the model is the dimension of each hidden layer. This data came from training on our length 10000 feature vectorization.

We found that a 1-layer neural net with a size 400 hidden layer performed equally as well or better than more complex methods with the tanh activation, and as it took less training time than the more complex models, we used this architecture to test different activation functions. We found that logistic and tanh activations had similar classification performance, but logistic took longer to train. ReLU didn't match up so well. We also tried a higher maximum number of training iterations on the highest performing architecture, but the results were unchanged.

Model	Acc.
400	0.867
400, LOGISTIC	0.867
400, 50, LOGISTIC	0.867
400, 100	0.866
800	0.866
400, 50	0.863
400, 200	0.862
200	0.862
200, 100	0.860
100	0.858
400, RELU	0.855

Table 2: Neural network 5-fold CV accuracy.

We also tried a very simple boosting method for less common classes, where we just copied each example of the least representative classes. It improved our cross-validation accuracy to .872, but generalized poorly, indicating overfitting.

3 Discussion

Our first set of features had relatively good performance, but was clunky to work with, as it was a very high-dimensional sparse matrix. This drove the development of smaller feature sets ($m = 10^4$ rather than 10^6), which made it much easier to iterate on models quickly. Indeed, we weren't even able to run some classifiers on a standard Mac laptop with the first feature set.

For model evaluation, we began with a baseline of an SVM with straightforward bag-of-words features and stochastic gradient descent optimization.

From there, we tried a Passive Aggressive classifier (Ethan, maybe you can elaborate a bit more?) which didn't do very well.

Iterating on the basic SVM, we were able to improve with a new version using the Huber loss function. We got a significant (about 2.5 percent) accuracy improvement from this switch. The Huber loss function is likely an improvement because..... not sure.

Even training on a 4-gram feature set, though, the SVM seemed to be hitting a limit, so from there we tried an LSTM neural net and a random forest classifier, which both led to slight improvements.

The fact that we were held around 80 percent accuracy regardless of some reasonable varied model choices most likely means that if we wanted to make substantial gains, we'd need to do more feature work, like with more domain-specific knowledge or at least educated guessing.