CS 181 Machine Learning
Practical 1 Report, Team *Advanced Representation*

Jing Wen[1], (Jeremiah) Zhe Liu[1], and (Vivian) Wenwan Yang[2]

[1]Department of Biostatistics, Harvard School of Public Health
[2]Department of Computer Science, Harvard School of Engineering and Applied Sciences

March 6, 2015

1 Exploratory Analysis

The training data set features the malware class (14 discrete categories) and XML behavior log regarding 3086 execurables.
The empirical distribution of malware classes is displayed in Figure 1. As shown, majority of malwares are of relatively low frequency ($< 2\%$), except `Swizzor` (542, 17.56%), `VB` (376, 12.18%), and `Agent` (114, 3.69%).

Agent	AutoRun	FraudLoad	FraudPack	Hupigon	Krap	Lipler	
114	50	37	32	41	39	53	
Magania	None	Poison	Swizzor	Tdss	VB	Virut	Zbot
41	1609	21	542	32	376	59	40

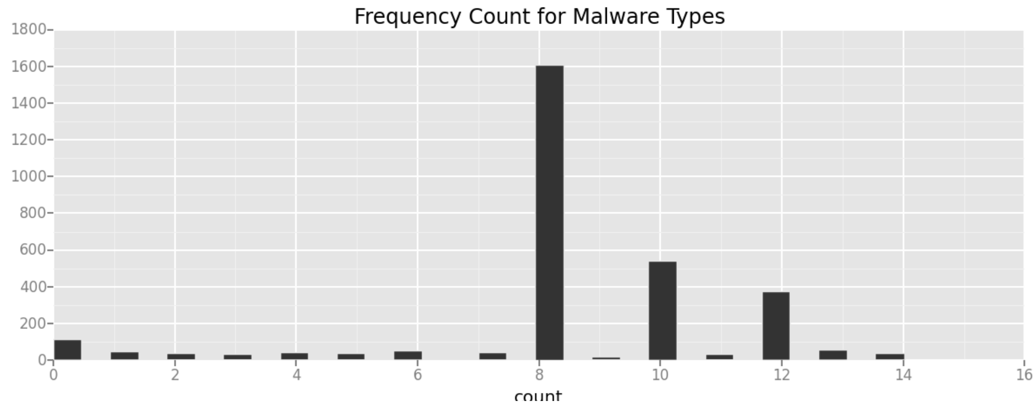

Figure 1: Emprical Count of Malware Types

The Input space is comprised XML behavior logs of the executables, which are generated from sand-box type environment where all system calls (as well as associated parameters and outcome status) of the executable are intercepted and reported in a structured XML file. Due to the complicated nature of the input space (structured strings), in the next section we discuss possible representation of such space and how to embed it into a euclidean/categorical feature space.

2 Embedding and Feature Extration

2.1 Representaion of Extraction Target

```
<load_image \
  filename="c:\342c547b28e9517f6fcf6c703933c0d9.EX" \
  successful="1" address="&#x24;400000" \
  end_address="&#x24;414000" size="81920" \
  filename_hash="hash_error"/>
```

Each XML log is consisted of a ordered set of structured report of system calls. As shown above, each call is consisted of a **operation type** (e.g. `load_image`), and **argument blocks** (e.g. `filename, successful, end_address, filename_hash`).

We currently consider information contained in such XML file as a ordered set of python dictionaries. To reasonably represent such information in the form of binary variables, Trinius et al (2009) argued that the parameters in each call can be stratified into a collection categorical variables of different levels of importance. For example, the supposedly most informative information for a system call, i.e. operation type, will be contained in level 1. Some less varying argument blocks (e.g. the extension of loaded dll in l̂oad_dll) to be contained in level 2, and the more varying, call-specific (thus less informative) argument blocks to be contained in level 2 and beyond. This is the so-called Malware Instruction Set (MIST) representation.

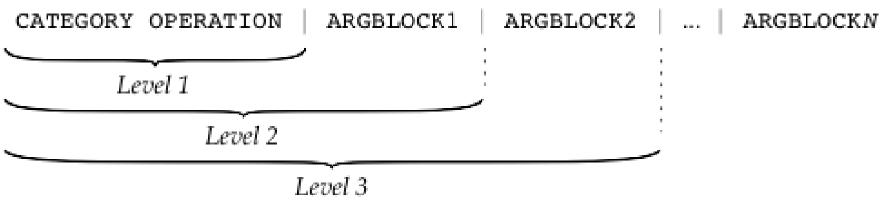
CATEGORY OPERATION | ARGBLOCK1 | ARGBLOCK2 | ... | ARGBLOCKN
Level 1
Level 2
Level 3

Figure 2: Malware Instruction Set (MIST) representation of XML records

However, to extract maximum about of information on executable behavior. We aim to represent both the information contained in <u>each call</u> and also <u>the order</u> of the system call. Since the call-specific features can be reasonably represented using the MIST idea, we aim to represent the order by considering a matrix representation of the order of system calls.

Specifically, for example, if we assume there are in total five types calls for all executables $[A, B, C, D, E]$ (say, A = `load_dll`), with the actual order of calls in a specific XML log to be $[A, B, C, D, E, B, C, D, A]$, we consider such a binary matrix:

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \\ B \\ C \\ D \\ A \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \\ & 1 & & 1 & \\ & & 1 & 1 & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix}$$

As shown, such matrix encodes the information of the order and types of system calls. Since in the training set there are in total 102 types of system calls for all executables, and typical length of a XML log is around 1396.52 . We are typically considering a extremely sparse matrix of dimension 1400 × 102, which is not extremely computationally impossible in terms of feature extraction. In the following paragraph, we denote such matrix as "**order matrix**".

2.2 Feature Extraction and Prunning

With the representation of XML logs as discussed in previous section, we are now ready to extract features for both the order-specific and call-specific information.

For **order-specific** features, we considered <u>total number of system calls</u> (the number of rows of order matrix), <u>types of first/last call</u> (the coordinate of the first/last rows of order matrix), and also the five

largest eigenvalues of the order matrix (in the context of dimension reduction, the set of eigenvalues of a matrix is the coordinates of this matrix in its principle axis).
For **call-specific** features, we considered the frequency of each types of system call in each file, which is refered as MIST level 1 in Trinius et al (2009), and the frequency of each type of level 2 arguments for each operation types as specified by the MIST.

Category	Feature
Order-specific Information	5 largest eigenvalues of order matrix Type of first call Type of last call Total number of call
Call-specific Information	MIST level 1 (**operation type**) MIST level 2

As a result, we extracted total number of 5561 features, which far exceeds the sample size. We thus performed feature prunning by removing categorical features who is non-zero for only 1 files, and obtained 752 features for the formal analysis.

3 Model Choice and Justifications

Based on observations from previous section, we realize that current learning task possesses below characteristics:

1. Moderate Sample Size
2. High dimensional Input Space
3. Complicated and unknown Feature-Outcome relation
4. Prediction as the Learning Goal

Based on above characteristics, we decided against using **regularized linear models** due to the difficulty in properly model the category-specific softmax using linear models. We also decided against using **Kernel-based methods** due to the large sample size. (This is because instead of working with the design matrix, Kernel methods working with a $n \times n$ Kernel matrix of the form $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$, where $\mathbf{\Phi}$ is a $n \times p$ kernel-transformed matrix of input features. Applying kernel matrix in current scenario implies working with a 3086 × 3086 potentially non-sparse matrix, whose computation cost is prohibitive for Cross-Validation based estimation). We also chose not to use **Bayesian Methods** due to our lack of prior knowledge with respect to the effect of each features. As a result, Bayesian regression doesn't seem to provide any benefit over regular models.
We decided to use **Random Forest** as our algorithm of choice. Introduced by Breiman and Cutler in 2001, random forest is essentially a ensemble method for regression trees. It offers a relatively scalable algorithm which is robust against noisy observations / covariates and powerful in capturing complex interaction between features. This method is traditionally considered having superb performance in terms of prediction accuracy.

3.1 Technical Detail of Random Forest

In this practical, we adopted the implementation `RandomForestClassifier` from `scikit-learn 0.15.1`, which offers a paralleled implementation of the original algorithm described in Breiman (2001). We describe the pseudo code for this algorithm in **Appendix**, and discuss in this section several important parameters in the algorithm:

B (Number of Trees in Forest).
One of the important features of random forest is it does not overfit, i.e. increasing the number of trees in forest does not overfit the data. Indeed, random forest estimate approximate the expectation of "real tree" conditional on the space of bootstrape samples $\hat{f}_{rf}(x) = E_{\Theta(\mathbf{Z})}T(x|\Theta(\mathbf{Z})) = \lim_{B\to\infty} \hat{f}(x)^B_{rf}$ (Hastie et al, 2009). Increase B will lead to a less biased and less noisy approximation of such value.

m (Number of Sampled Features) & $\mathbf{d_{max}}$ **(Maximum Tree Depth)**.
Although increase **B** will only bring estimator toward a constant limit, this limit itself, however, may overfit depending on d_{max} and m. This is because intuitively, too deep a tree indicates a overly rich model, and too many considered feature increase correlation between generated trees. Both may incur unnecessary variance. It is thus crucial to perform parameter selection for **m** and $\mathbf{d_{max}}$ during model fitting.

4 Parameter Selection

In the next step, we consider the effect of **m** and d_{max} in our prediction performance through 10-fold cross validation. With the misclassification rate as the outcome metric, we seek to search over a grid of (**m**, d_{max}) since the convexity of the current problem is unclear. For regression problems, Breiman recommended setting $\mathbf{m} = \frac{p}{3}$ and node size equal to 5, which equals to $\mathbf{m} = 752/3 \approx 250$ and $d_{max} = \log_2(3086/5) \approx 10$. In current setup, however, smaller **m** usually works better due to the concern of overfitting. We hence decide to search over the grid $\left[\mathbf{m} \in \{30, 32, ..., 124\}\right] \times \left[d_{max} \in \{16, ..., 30\}\right]$ so that it covers a reasonable range of the theoretically sound values. The misclassification rate surface over candidate parameter values are visualized in Figure 3 and in Table attached to the end of this file. As a conclusion, we selected $(\mathbf{m} = 40, d_{max} = 28)$. With the selected parameters, we ran a random forest with size **B** = 5000 in order to achieve closest approximation toward the theoretical limit $E_{\Theta(\mathbf{Z})}T(x|\Theta(\mathbf{Z}))$ within reasonable machine computing time (2 hours). The final 10-fold CV AME within training dataset is calculated to be 0.0097.

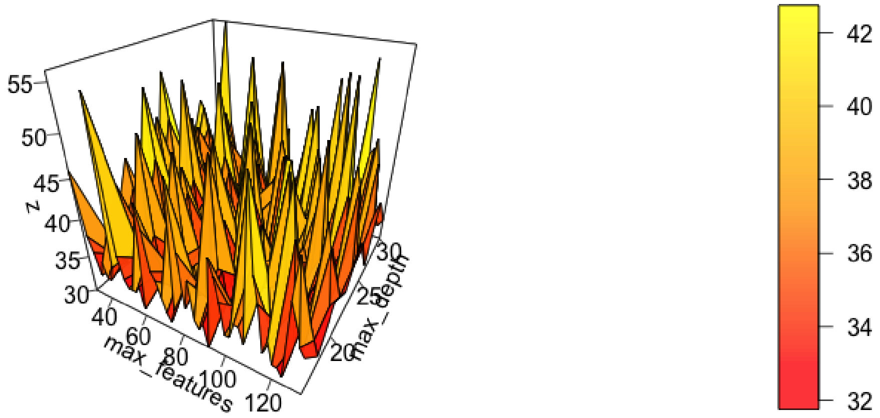

Figure 3: Malware Instruction Set (MIST) representation of XML records

5 Discussion

Given the pre-extracted feature provided by this dataset, authors believe that our current model should offer best possible performance in terms of prediction. However, prediction performance may be further improved by extracting more energy-relevant, and preferably continous features from the order matrix, and explore feature-outcome relationship using again random forest to achieve a more satisfactory prediction result. Unfortunately, limited by the time and computing environment available to authors, above procedure was not carried out.

Reference

1. L Breiman, Random Forests, Machine Learning, 45(1), 5-32, 2001.
2. T Hastie. R Tibshirani. J Friedman, ”Elements of Statistical Learning”, Springer, 2009.
3. P Trinius et al, ”A Malware Instruction Set for Behavior-Based Analysis”, Technical Report, University of Mannheim, Germany, 2009

Appendix: Algorithm for Random Forest

Input: $\mathbf{Z} = (\mathbf{t}_{N\times1}, \mathbf{X}_{N\times p})$

Parameter:

- B: The number of trees in the forest
- m: The number of features to sample when splitng nodes.
- d_{max}: The maximal allowed depth of each tree.
- n_{min}: The minimal allowed size the of the splitted node.

Algorithm:

```
1. For b = 1 to B
      (a) Draw a bootstrap sample $\mathbf{Z}^*$ of $\mathbf{Z}$
      (b) (Grow a regression tree $T_b$ using $\mathbf{Z}^*$)
          Untile maximal depth of tree ($d_{max}$)/minimal size of node ($n_{min}$) is reached, for every node in current tree:
            i. Randomly sample m features from the p features
           ii. Find optimal split (in the sense of minimizing RMSE) of current node with respect to select features.
          iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}$ $b \in \{1, ..., B\}$.
3. Predict for a new point $\mathbf{x}$ through majority vote.
```

n_feature/depth	16	18	20	22	24	26	28	30
30	0.0149	0.0110	0.0143	0.0104	0.0169	0.0120	0.0149	0.0113
32	0.0123	0.0104	0.0130	0.0107	0.0149	0.0117	0.0146	0.0107
34	0.0117	0.0107	0.0120	0.0113	0.0117	0.0120	0.0136	0.0107
36	0.0107	0.0104	0.0120	0.0113	0.0113	0.0104	0.0110	0.0117
38	0.0113	0.0139	0.0100	0.0156	0.0113	0.0143	0.0120	0.0181
40	0.0107	0.0123	0.0107	0.0133	0.0107	0.0117	0.0097	0.0136
42	0.0107	0.0113	0.0104	0.0123	0.0107	0.0126	0.0107	0.0126
44	0.0113	0.0113	0.0107	0.0113	0.0117	0.0110	0.0110	0.0117
46	0.0178	0.0113	0.0172	0.0113	0.0152	0.0104	0.0146	0.0110
48	0.0120	0.0110	0.0146	0.0100	0.0136	0.0107	0.0133	0.0117
50	0.0110	0.0107	0.0113	0.0107	0.0117	0.0104	0.0113	0.0113
52	0.0107	0.0107	0.0107	0.0113	0.0113	0.0110	0.0107	0.0104
54	0.0107	0.0162	0.0104	0.0172	0.0100	0.0133	0.0110	0.0149
56	0.0107	0.0123	0.0107	0.0133	0.0104	0.0146	0.0107	0.0123
58	0.0110	0.0107	0.0110	0.0104	0.0104	0.0117	0.0110	0.0120
60	0.0100	0.0107	0.0113	0.0123	0.0110	0.0120	0.0110	0.0120
62	0.0146	0.0100	0.0146	0.0110	0.0169	0.0113	0.0172	0.0113
64	0.0136	0.0110	0.0136	0.0107	0.0139	0.0110	0.0136	0.0100
66	0.0120	0.0110	0.0110	0.0104	0.0133	0.0104	0.0107	0.0107
68	0.0117	0.0104	0.0113	0.0113	0.0113	0.0110	0.0117	0.0107
70	0.0110	0.0159	0.0104	0.0143	0.0113	0.0159	0.0107	0.0162
72	0.0104	0.0120	0.0117	0.0120	0.0113	0.0143	0.0113	0.0136
74	0.0100	0.0113	0.0110	0.0113	0.0107	0.0133	0.0107	0.0123
76	0.0107	0.0117	0.0113	0.0113	0.0100	0.0113	0.0104	0.0139
78	0.0159	0.0104	0.0159	0.0120	0.0152	0.0107	0.0172	0.0113
80	0.0133	0.0107	0.0126	0.0100	0.0139	0.0110	0.0113	0.0113
82	0.0110	0.0120	0.0130	0.0100	0.0126	0.0126	0.0133	0.0104
84	0.0123	0.0104	0.0100	0.0117	0.0120	0.0120	0.0120	0.0110
86	0.0107	0.0162	0.0110	0.0162	0.0107	0.0146	0.0110	0.0133
88	0.0104	0.0120	0.0104	0.0133	0.0104	0.0139	0.0107	0.0130
90	0.0107	0.0120	0.0107	0.0104	0.0107	0.0136	0.0117	0.0117
92	0.0097	0.0107	0.0100	0.0123	0.0100	0.0117	0.0107	0.0126
94	0.0136	0.0107	0.0149	0.0104	0.0146	0.0120	0.0156	0.0117
96	0.0130	0.0100	0.0130	0.0104	0.0113	0.0113	0.0139	0.0107
98	0.0120	0.0107	0.0120	0.0104	0.0123	0.0113	0.0123	0.0100
100	0.0107	0.0110	0.0133	0.0107	0.0117	0.0120	0.0120	0.0104
102	0.0107	0.0159	0.0110	0.0152	0.0110	0.0162	0.0117	0.0165
104	0.0113	0.0130	0.0120	0.0152	0.0107	0.0146	0.0113	0.0130
106	0.0113	0.0120	0.0117	0.0143	0.0123	0.0126	0.0117	0.0123
108	0.0107	0.0120	0.0110	0.0117	0.0123	0.0126	0.0110	0.0110
110	0.0169	0.0126	0.0156	0.0107	0.0149	0.0110	0.0172	0.0110
112	0.0136	0.0104	0.0120	0.0120	0.0133	0.0110	0.0123	0.0113
114	0.0126	0.0107	0.0123	0.0113	0.0117	0.0107	0.0126	0.0113
116	0.0107	0.0107	0.0104	0.0110	0.0110	0.0107	0.0113	0.0110
118	0.0113	0.0162	0.0110	0.0162	0.0107	0.0175	0.0120	0.0175
120	0.0104	0.0143	0.0107	0.0117	0.0113	0.0130	0.0117	0.0136
122	0.0104	0.0100	0.0113	0.0113	0.0107	0.0107	0.0113	0.0139
124	0.0100	0.0107	0.0117	0.0113	0.0104	0.0107	0.0104	0.0110