## Design of ML Maldoc Classifier

In this session, we focus on designing binary classifiers, a particular kind of learning systems, which classify new data into two predefined categories. Classifiers usually make predictions by computing some numeric or probabilistic score and comparing it with a fixed threshold. We focus on the following aspects of classifier design:

1. Dataset
2. Feature Engineering
3. Classification Alg. Selection
4. General ML Framework
5. Model Updates

### Dataset

Our dataset, in a total of **201368** PDF samples, can be divided into two classes: **28332** benign and **173036** malicious samples.

Among those, **156035** malicious samples are downloaded from VirusShare; **9000** malicious samples are from the Contagio dataset, the rest are obtained from two popular search engines. Besides, we obtained the open source dataset, in the present of feature vectors, collected for PDFRate[4] evaluation. This dataset contains 20,000 balanced samples, with 5,000 benign and 5,000 malicious samples from Contagio dataset, and 5,000 benign samples obtained from Google as well as 5,000 malicious samples from VirusTotal.

Further, we randomly select around **2000** malicious documents from our dataset to generate around **10,000** adversarial samples used targeting deployed ML Model for model evasion attack in Session 4.

(**TODO: Plot Dataset Figure diversity -> by malware family; timeliness -> by year**)

### Feature Engineering

We focus on parsing the structure and metadata of these files and select features manually. Significant amount of work in feature engineering reveal that the combination of structure and meta based features perform well enough in terms of prediction accuracy, even if we are not heavily focus on analyzing the content and JavaScript of the file. As shown in Table 2, we apply some basic statistics in calculating the average value of each feature and filter out some representative features which the average values between benign and malicious samples are significantly different.

Features such as count\_font and count\_box: There are several objects like font, box contained in the benign samples as PDF file mainly uses these objects for description. However, malicious files do not aim at describing information, instead they run the malicious code embedded in the file to launch the attack.

Features such as count\_page\_obj and count\_obj: Generally, obj in benign files are many more than those in malicious files. When calculating the number of obj in the same page, that in a malicious file is twice as many as that in a benign file. Thus, if the number of obj in the same page increases sharply, the file is likely to be malicious.

Features such as count\_endobj and count\_endstream: In benign files, the endobj refers to the end of an object. Yet a maldoc seldom contains endobj and endstream, for which it aims at confusing the parser to make it fail to obtain the whole object when parsing the malicious file, or fail to parse the malicious documents which can then evade detection successfully later. We call this the model evasion attack.

Features such as count\_js: The main tactic of the malicious document is to embed JS code in the file to execute malicious behaviors. In this way, JS codes contained in a maldoc are generally on average, much more than those in a benign file.

Features such as count\_acroform\_obs: AcroForm is introduced in PDF Specification 1.2, which is to collect information from users via interaction. The form can display, capture and edit the data, etc. Moreover, it can conduct dynamic interaction from the interactive and editable forms which contain characteristics like dynamic calculation, verification and so on, to the forms generated by servers and filled in by machine. With those characteristics, the form is vulnerable to obscuration and encryption by the attacker. As a typical document being exploited, the value of AcroForm in a malicious sample usually doubles than that of a benign sample.

Table 2: Average Value of Features between Benign and Malicious Samples

|  |  |  |
| --- | --- | --- |
| Feature | Benign | Malware |
| **count\_font** | **14.64** | **0.55** |
| **count\_acroform\_obj** | **700** | **1400** |
| **count\_box\_a4** | **12001** | **200** |
| **count\_box\_legal** | **395040** | **0** |
| count\_box\_letter | 7291529 | 866773 |
| count\_box\_other | 32.18 | 1.74 |
| count\_box\_overlap | 1000 | 0 |
| **count\_endobj** | **95.80** | **9.68** |
| **count\_endstream** | **30.43** | **3.78** |
| **count\_page\_obj** | **8001** | **16003** |
| count\_image\_large | 110711 | 400 |
| count\_image\_med | 465247 | 6401 |
| **count\_image\_small** | **915892** | **12002** |
| count\_image\_total | 36.56 | 0.30 |
| count\_image\_xlarge | 300 | 0 |
| count\_image\_xsmall | 21.64 | 0.11 |
| **count\_js** | **0.71** | **1.01** |
| **count\_obj** | **100.96** | **12.01** |
| count\_objstm | 1.57 | 0.15 |

### Classification Alg. Selection

We classify the collected files by extracted features which are used as the testing dataset. We select several algorithms including Decision Tree, Random Forest and SVM for comparison. As shown in Table 3, Random forest are known for good efficiency - with minute level training time and ms level prediction time, good effectiveness - with accuracy as high as 99%, excellent robustness (detail in Section 4), easy to interpret and good generalization.

The output of random forest is essentially an ensemble of a multitude of decision trees. That said, Random forest is an ensemble classifier applying the technique of bagging training data. Each node in a decision tree is constructed based on a randomly selected subset of features, as well as the best split at each node, which is determined by training data for that node. Finally, the classification result is determined by the votes of each tree.

We only employs random forest from then on.

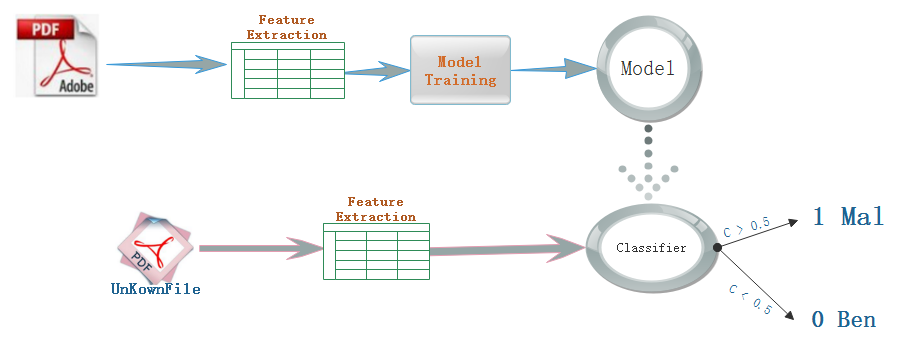
Table 3 Comparison of Different ML Models

|  |  |  |  |
| --- | --- | --- | --- |
|  | SVM | Decision Tree | Random Forest |
| Accuracy | 75.23% | 82.41% | 99.64% |
| Training Time (the whole dataset) | 58m18s | 4s | 56s |
| Prediction Time (for each sample) | 1.2ms | 0.1ms | 0.1ms |

### General ML Framework

The proposed ML framework are depicted in Figure 1. Our goal is to train a model for maldoc detection. Firstly, we need to collect a great amount of malicious and benign documents during the data collection phase. Secondly, we have manually designed and extracted hundreds of representative features from each documents during our feature engineering phase, in the hope that each feature vector can represent the document nicely. Finally, we have trained the ML model so that the model can fit the underlying training data distribution well. At this point, our model is ready for serving & prediction. When a new sample is presented to the model, it can return a confidence score to predict whether the sample is malicious or not. Figure 1 provides a good description of model training and prediction.

Figure 1 General ML Framework (TODO: FIX)



1. **Model Updates**

We have 2 major updates for our model during experiment and each model provides a probabilistic estimate of the PDF’s maliciousness. All 3 classifiers deployed by us produce as their result the output of their decision function, i.e., a real value in the interval [0,1] denoting the percentage of decision that have labeled the submitted file as malicious. We apply the default value of threshold (0.5) when prediction.

**Model 1**: Use peepdf (<https://github.com/jesparza/peepdf>)as the external parser for feature extraction. After computation and quantization, these features can be used for training and prediction. We extract 133 features which contain static attributes of structure such as count\_font, size and count\_startxref, content such as title\_oth and subject\_lc, metadata such as producer\_oth and producer\_len. But the limitation for peepdf is obvious: Although we enjoy high accuracy in Model 1, half of the PDF files can NOT be parsed correctly by the external parser. The main reasons are defected file structure or intended file obfuscation technique.

**Model 2**: In this model, in order to conquer the major deficiency for model 1, we switch to a much more robust external parser in the mimicus (<https://github.com/srndic/mimicus>) framework. By using this new external parser, nearly all the PDF files can be properly parsed. For the training of Model 2, we initially use the balanced dataset for training and testing. This balanced dataset includes 20,000 malicious and 20,000 benign examples selected randomly from the whole dataset. Besides, we extract 135 features for Model 2 in total. The main algorithm for model is random forest. After grid search and model parameter tuning, the accuracy of Model 2 increases up to 99.99%, with a false positive rate being 0.012%. We serve our models to major commercial cloud service providers for Model-as-a-Service.

**Model 3:** The big differences between Model 2 and Model 3 is its robustness. Model 2 has a assumption that during model serving & prediction, a benign working environment is provided. While in Model 3, we assume adversaries to be presented and there are high chances that adversarial examples will be submitted for model prediction. We will discuss model evasion attack in detail in Session 4 and propose a few effective defense strategies.