Learning to Classify PC Malwares

Jingxiang Li   Advisor: Yuhong Yang

School of Statistics, UMN

Outline

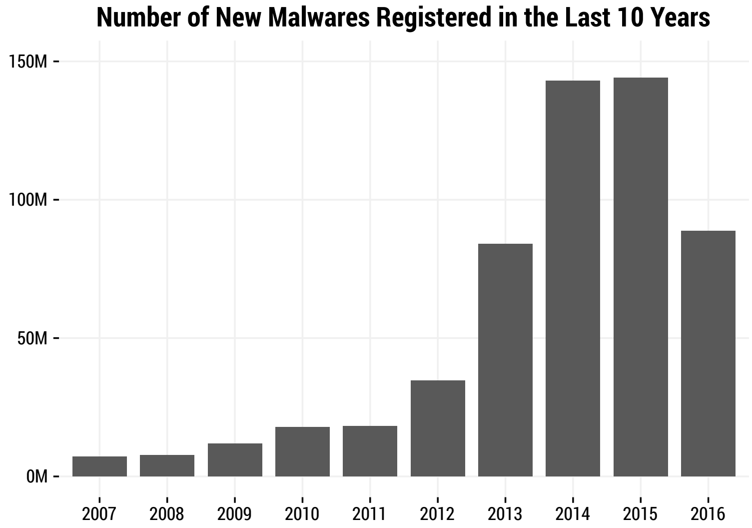
- Introduction
- Background
- Dataset Description
- Feature Engineering
- Predictive Modeling
- Model Evaluation
- Experiment Result

Introduction

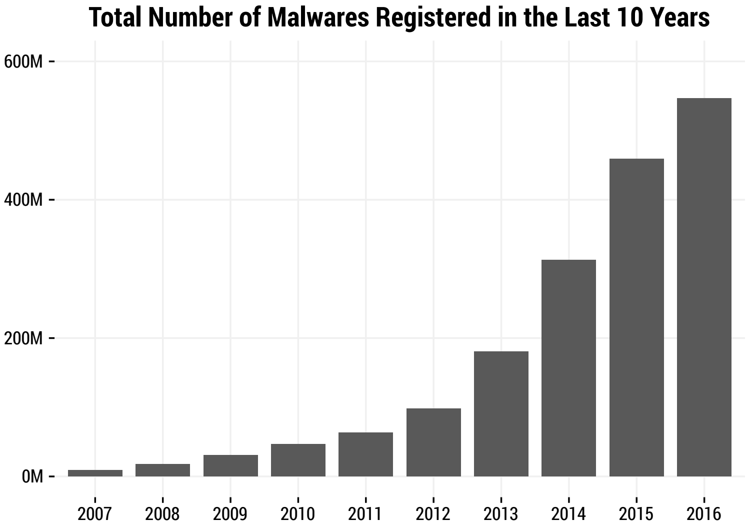
Malware

- Short for “malicious software”
- Disrupt computer operations
- Gather sensitive information
- Gain access to private computer systems
- Display unwanted advertising.

Introduction



Introduction



Introduction

Malware Industry

- Malware industry has become a well organized market involving large amounts of money.
- “Malware as a Service”
  - 400,000 Users and Organizations Globally
  - Anyone can obtain a network controlling 10,000 compromised computers for just $1,000.

Introduction

Evasion

- Since the beginning of 2015, a sizable portion of malware utilizes a combination of many techniques designed to avoid detection and analysis.
- Polymorphism is introduced to malwares. Malicious files belonging to the same family looks like totally different files.
- Malware classification has become a challenge.

Goal of the Study

Malware Classification

- Given a malware executable, determine which family it belongs to.
- Build a statistical learning based framework to automatically classify malware.

Background

How are Executables Generated

```
gcc -o helloworld helloworld.c
```

1. Write code in a programming language
2. Compiler generates assembly code version
3. Assembler converts assembly code into binary object file
4. Linker merges object files and libraries together and generates the executable

Background

Representations of an Executable

- Original code in programming language
- Assembly code generated by compiler
- Binary object file generated by assembler and linker

Background

Representations of an Executable

- Original code in programming language ✗
- Assembly code generated by compiler ✓
- Binary object file generated by assembler and linker ✓

Assembly and Binary code can be obtained by Interactive Disassembler (IDA)

Background

Example of Assembly Code

```
.text:00401090 8B 44 24 10    mov     eax, [esp+10h]
.text:00401094 8B 4C 24 0C    mov     ecx, [esp+0Ch]
.text:00401098 8B 54 24 08    mov     edx, [esp+8]
.text:0040109C 56             push    esi
.text:0040109D 8B 74 24 08    mov     esi, [esp+8]

......

.idata:005265CC               ; Imports from WS2_32.dll
.idata:005265CC               ;
.idata:005265D0               ; int __stdcall WSACleanup()
.idata:005265D0 ?? ?? ?? ??   extrn WSACleanup:dword ;
                              CODE XREF: _WinMain@16_0+
.idata:005265D0               ; DATA XREF: _WinMain@16_0+
```

Background

Information in Assembly Code

- Code segments (e.g. .text, .idata) ✓
- Memory address of the line of code (e.g. 00401090)
- Binary representation of the code (e.g. 8B 44 24 10) ✓
- Operation code (Opcode), single instruction executed by the CPU (e.g. mov, push) ✓
- Operands (e.g. eax, [esp+10h])
- Dynamic link libraries imported to the program (e.g. WS2_32.dll)
- Function called by the program (e.g. WSACleanup)

Dataset Description

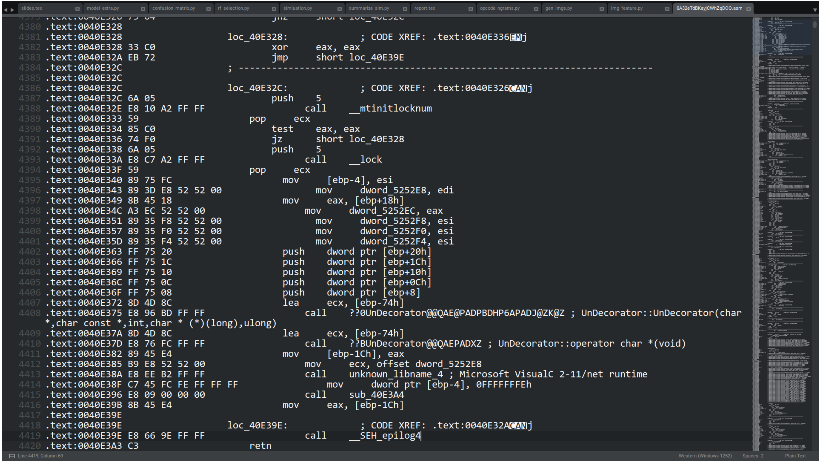
Dataset

- From Microsoft
- 10,826 malwares, all labeled
- 8 malware families
- 185 GB
- 2 text files for each malware variant

Table 1: Malware Families

Family	Count
Ramnit	1541
Lollipop	2478
Kelihos_ver3	2942
Vundo	475
Tracur	751
Kelihos_ver1	398
Obfuscator.ACY	1228
Gatak	1013

Dataset



```
.text:0040E328
.text:0040E328                 loc_40E328:             ; CODE XREF: .text:0040E336EMj
.text:0040E328 33 C0                   xor     eax, eax
.text:0040E32A EB 72                   jmp     short loc_40E39E
.text:0040E32C                 ; ---------------------------------------------------------------------------
.text:0040E32C
.text:0040E32C                 loc_40E32C:             ; CODE XREF: .text:0040E326CANj
.text:0040E32C 6A 05                   push    5
.text:0040E32E E8 10 A2 FF FF                  call    __mtinitlocknum
.text:0040E333 59                  pop     ecx
.text:0040E334 85 C0                   test    eax, eax
.text:0040E336 74 F0                   jz      short loc_40E328
.text:0040E338 6A 05                   push    5
.text:0040E33A E8 C7 A2 FF FF                  call    __lock
.text:0040E33F 59                  pop     ecx
.text:0040E340 89 75 FC                  mov     [ebp-4], esi
.text:0040E343 89 3D E8 52 52 00               mov     dword_5252E8, edi
.text:0040E349 8B 45 18                  mov     eax, [ebp+18h]
.text:0040E34C A3 EC 52 52 00                  mov     dword_5252EC, eax
.text:0040E351 89 35 F8 52 52 00               mov     dword_5252F8, esi
.text:0040E357 89 35 F0 52 52 00               mov     dword_5252F0, esi
.text:0040E35D 89 35 F4 52 52 00               mov     dword_5252F4, esi
.text:0040E363 FF 75 20                  push    dword ptr [ebp+20h]
.text:0040E366 FF 75 1C                  push    dword ptr [ebp+1Ch]
.text:0040E369 FF 75 10                  push    dword ptr [ebp+10h]
.text:0040E36C FF 75 0C                  push    dword ptr [ebp+0Ch]
.text:0040E36F FF 75 08                  push    dword ptr [ebp+8]
.text:0040E372 8D 4D 8C                  lea     ecx, [ebp-74h]
.text:0040E375 E8 96 BD FF FF                  call    ??0UnDecorator@@QAE@PADPBDHP6APADJ@ZK@Z ; UnDecorator::UnDecorator(char *,char const *,int,char * (*)(long),ulong)
.text:0040E37A 8D 4D 8C                  lea     ecx, [ebp-74h]
.text:0040E37D E8 76 FC FF FF                  call    ??BUnDecorator@@QAEPADXZ ; UnDecorator::operator char *(void)
.text:0040E382 89 45 E4                  mov     [ebp-1Ch], eax
.text:0040E385 B9 E8 52 52 00                  mov     ecx, offset dword_5252E8
.text:0040E38A E8 EE B2 FF FF                  call    unknown_libname_4 ; Microsoft VisualC 2-11/net runtime
.text:0040E38F C7 45 FC FE FF FF FF                mov     dword ptr [ebp-4], 0FFFFFFFEh
.text:0040E396 E8 09 00 00 00                  call    sub_40E3A4
.text:0040E39B 8B 45 E4                  mov     eax, [ebp-1Ch]
.text:0040E39E
.text:0040E39E                 loc_40E39E:             ; CODE XREF: .text:0040E32ACANj
.text:0040E39E E8 66 9E FF FF                  call    __SEH_epilog4
.text:0040E3A3 C3                  retn
```

Figure 1: .asm File

Dataset

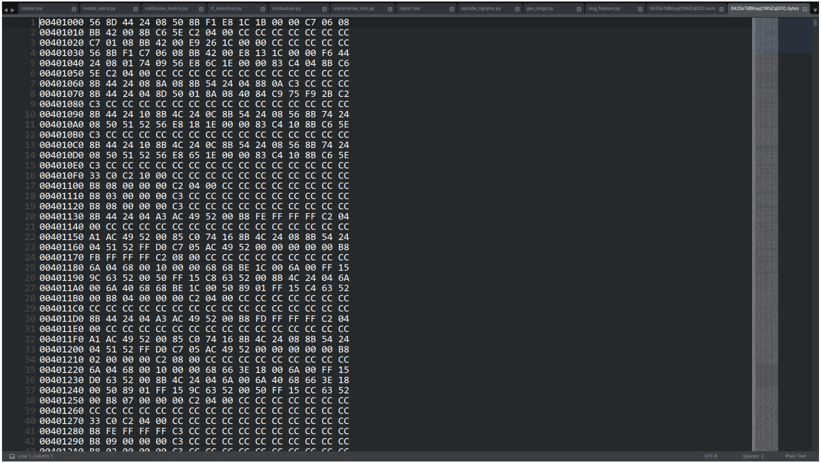


Figure 2: .bytes File

Feature Engineering

Source of Features

- Text features from ".asm" file
- Text features from ".bytes" file
- Image features from ".bytes" file
- Meta data features from both ".asm" file and ".bytes" file

Feature Engineering

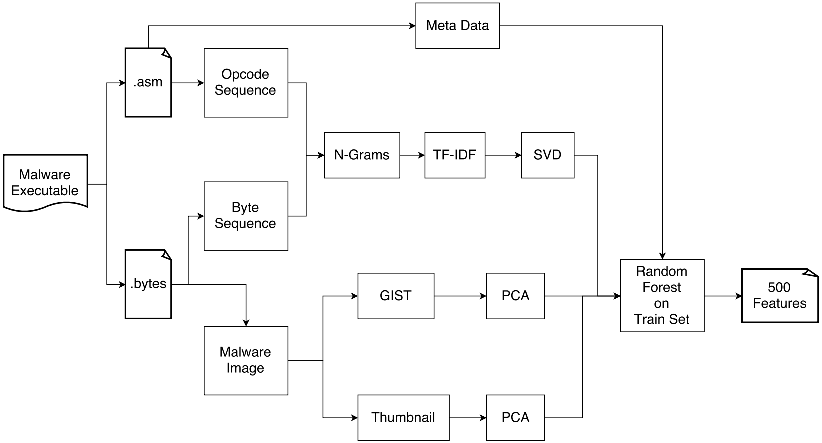


Figure 3: The Feature Engineering Process

Feature Engineering

Text Features from ".asm" File

1. Extract Opcode sequence
2. Count n-grams (n = 1, 2, 3, 4) on Opcode sequence
3. TF-IDF to vectorize and re-weight n-grams
4. SVD for Latent Semantic Analysis (LSA) and dimensionality reduction

Text Features from “.asm” File

Extract Opcode sequence

```
.text:00401390 8B 4C 24 04         mov     ecx, [esp+arg_0]
.text:00401394 B8 1F CD 98 AE      mov     eax, 0AE98CD1Fh
.text:00401399 F7 E1               mul     ecx
.text:0040139B C1 EA 1E            shr     edx, 1Eh
.text:0040139E 69 D2 FA C9 D6 5D   imul    edx, 5DD6C9FAh
.text:004013A4 56                  push    esi
.text:004013A5 57                  push    edi
.text:004013A6 8B F9               mov     edi, ecx
```

Opcode Seq $\Rightarrow$ mov, mov, mul, shr, imul, push, push, mov

Text Features from “.asm” File

Count n-grams

- mov, mov, mul, shr, imul, push, push, mov
- 1-gram $\Rightarrow$ {mov: 3, push: 2, mul: 1, imul: 1, shr: 1}
- bi-grams $\Rightarrow$ {(mul, shr): 1, (push, push): 1, (push, mov): 1, (imul, push): 1, (mov, mul): 1, (mov, mov): 1, (shr, imul): 1}
- tri-grams $\Rightarrow$ {(shr, imul, push): 1, (mov, mov, mul): 1, (mov, mul, shr): 1, (mul, shr, imul): 1, (imul, push, push): 1, (push, push, mov): 1}

Text Features from ".asm" File

n-grams

- 658 1-gram, all
- 1524 bi-grams, > 150 times in at least one file
- 5225 tri-grams, > 100 times in at least one file
- 9130 4-grams, > 100 times in at least one file

Text Features from ".asm" File

TF-IDF

- Let t be the n-gram term, d be the document, D be the Corpus

$$\mathrm{TF}(t, d) = f_{t,d} \quad \mathrm{IDF}(t, D) = \log\left(1 + \frac{N}{|\{d \in D : t \in D\}|}\right)$$

$$\mathrm{TFIDF}(t, d, D) = \mathrm{TF}(t, d) \cdot (\mathrm{IDF}(t, D) + 1)$$

- Construct matrix, file as row, n-gram as column, TF-IDF as value
- Apply L2 normalize for each row (cosine distance)
- Why IDF? Tokens that occur very frequently in a given corpus are not informative

Text Features from ".asm" File

SVD

- Latent Semantic Analysis and Dimensionality Reduction
- $M_{m,n} = U_{m,m}\Sigma_{m,n}V^T_{n,n}$
- $M_{m,n} \approx U_{m,k}\Sigma_{k,k}V^T_{n,k}$ closest approximation with rank k
- $U_{m,k}$ the reduced version of M, with k Topic vectors
- Choose k, use theorem

$$||M||_F^2 = \sum \sigma_i^2$$

- 95% norm explained is fine

SVD

- 100 1-gram topics
- 200 2-gram topics
- 400 3-gram topics
- 500 4-gram topics
- Total from 16537 to 1200

Feature Engineering

Text Features from “.bytes” File

1. Extract Byte sequence
2. Count n-grams (n = 1, 2, 4) on Byte sequence
3. TF-IDF to vectorize and re-weight n-grams
4. SVD for Latent Semantic Analysis (LSA) and dimensionality reduction

Feature Engineering

Image Features From ".bytes" File

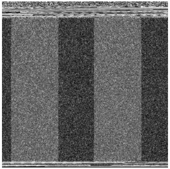
1. Visualize malware image and scale it
2. Obtain feature vector by GIST descriptor and PCA
3. Obtain feature vector by image thumbnail and PCA

Image Features From ".bytes" File

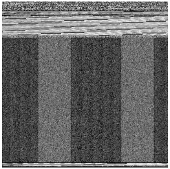
Malware Visualization

- From .bytes file, instructions are represented sequence of double hexadecimal points
- 00 to FF in hex is 0 to 255
- Use this value as a gray scale value
- Determine the width of image

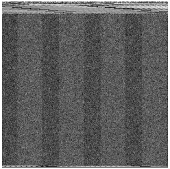
Image Features From ".bytes" File



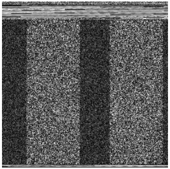
(a) 01IsoiSMh5gxyDYTl4CB



(b) 02zcUmKV16Lya5xqnPGB



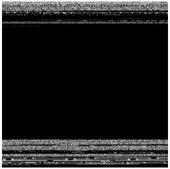
(c) 2ZNUOyAMv67lcYPoetkR



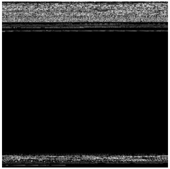
(d) 40ofYnDCl8TKJXwyGmrP

Figure 4: Sample Malware Images for Lollipop

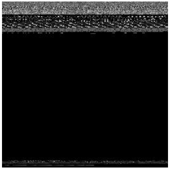
Image Features From “.bytes” File



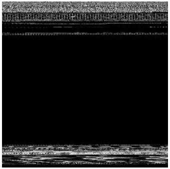
(a) 0AguvpOCcaf2myVDYFGb



(b) 2TaBCxc4msyVU45YDRuOd



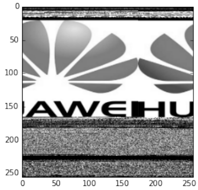
(c) 6jCs7bHULZXW3kKtxNRw



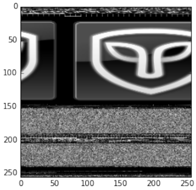
(d) 8lTjbp3rnwtLh104E57v

Figure 5: Sample Malware Images for Obfuscator.ACY

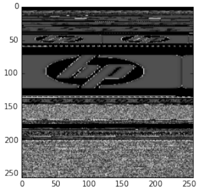
Image Features From ".bytes" File



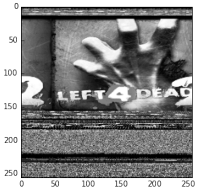
(a) 5764RnJYTirWq1utgdkN



(b) 2DBKbxPnVCyiLzqAHU9c



(c) NxrC3kQvsl5AZ4WtowHD



(d) goYPdUatw3BQEl6iCI0D

Figure 6: Fun Malware Image

Image Features From “.bytes” File

GIST

- Convolve the image with 20 pre-designed Gabor filters to construct 20 feature maps
- Divide each feature map into 16 regions (by a 4×4 grid)
- Average the feature values within each region
- Concatenate feature values together, length 320
- PCA for dimension reduction

Image Features From ".bytes" File

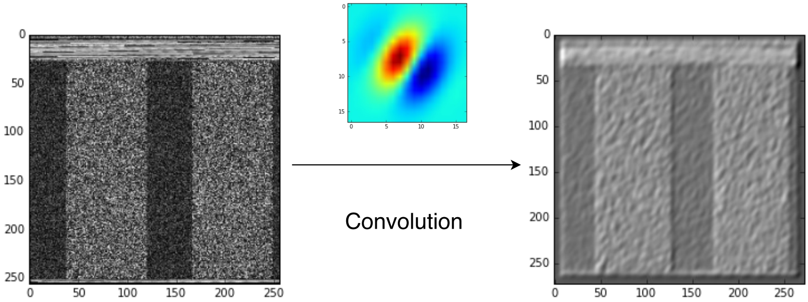


Figure 7: GIST

Image Features From “.bytes” File

Image Thumbnail

- Rescale the image to 32×32 and flatten it to a vector
- PCA for dimension reduction

Feature Engineering

Meta Data Features From Both ".asm" File and ".bytes" File

- Segment count from ".asm" file (.text, .data, .idata), TF-IDF, SVD
- File size for ".asm" file
- File size for ".bytes" file

Feature Engineering

Concatenation and Selection

- Concatenate all features together, more than 4,000
- Supervised feature selection by training Random Forest on the Training Set, 500 ultimate features

Feature Engineering

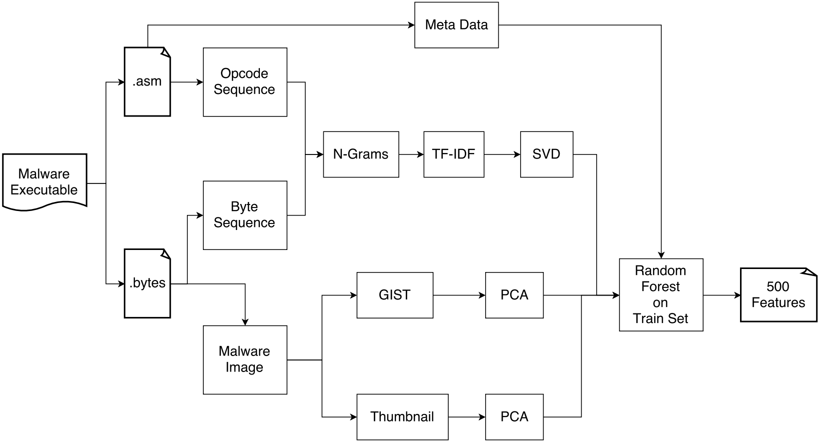


Figure 8: The Feature Engineering Process

Feature Engineering

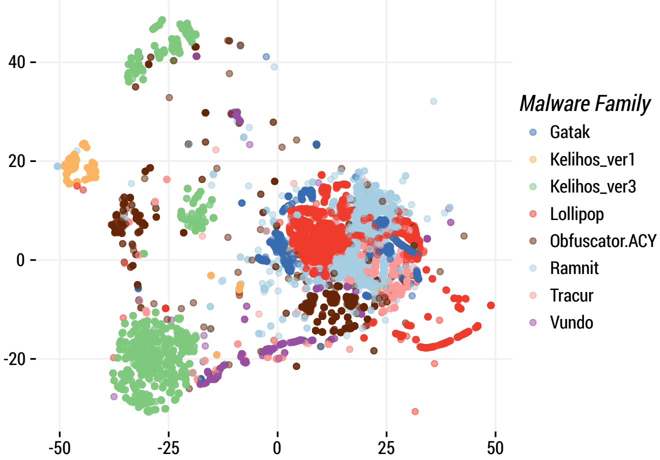


Figure 9: t-SNE Visualization with 500 Selected Features

Predictive Modeling

Candidates

- Generative Model
  - Naive Bayes (NB)
  - Linear Discriminant Analysis (LDA)
  - Quadratic Discriminant Analysis (QDA)
- Discriminative model
  - Multinomial Logistic Regression (Multi-Logreg)
  - One-vs-Rest Logistic Regression (OvR-Logreg)
  - Support Vector Machine with Probability Calibration (SVM)
  - Random Forest Classifier (RF)
  - Extremely randomized trees (ExtraTrees)
  - eXtreme Gradient Boosting (XGBoost)

Generative Model

Idea

- Estimate $P(\vec{x}|y)$ based on different assumptions
- Make prediction by Bayes' Rule

$$P(y|\vec{x}) = \frac{P(\vec{x}|y)p(y)}{P(\vec{x})}$$

Generative Model

Candidates

- NB, independent Gaussian: $P(\vec{x}|y) = P(x_1|y)P(x_2|y)\ldots$
- LDA, Multi-Gaussian, same Covariance Matrix across classes
- QDA, Multi-Gaussian, different Covariance Matrix

Discriminative Model

Idea

- Directly model $P(y|\vec{x})$
- Training by Maximum a Posteriori, or Maximum Likelihood

Discriminative Model

Logistic Regression

- Linear combination of features
- Softmax function
- Minimize cross entropy with one-hot encoding
- Multinomial and One-vs-Rest

Discriminative Model

Support Vector Machine

- Primal form

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \zeta_i + \lambda ||w||^2 \quad \text{s.t.} \quad y_i(x_i \cdot w + b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0$$

- Dual form

$$\max_{a_1,\dots,a_n} f(a_1 \dots a_n) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i a_i (x_i \cdot x_j) y_j a_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_i y_i = 0, \text{ and } 0 \leq a_i \leq \frac{1}{2n\lambda}$$

Discriminative Model

Support Vector Machine

- Kernel tricks for non-linear classification (Gaussian kernel)
- Probability calibration: Bining or softmax fitting

Discriminative Model

Random Forest

- Ensemble of classification trees
- Bootstrap sampling
- Randomly select features for each node split

Extremely Randomized Trees

- Randomly draw threshold for each node split

Discriminative Model

eXtreme Gradient Boosting

- Minimize a loss function (Logloss in this case)
- Additively minimize the loss function by adding classification trees
- The first and second order information is used in building each classification tree

Evaluation

Data Split

- 5% train set (542), 95% test set (10,248) by randomized stratified sampling.
- Enlarge the difference among classifiers
- More stable estimation of predictive metrics
- Cross-validation for hyper-parameter tunning
- Repeat the procedure for 10 times

Evaluation

Metrics

- Multi-class Logloss

$$\text{Logloss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

- Multi-class Accuracy

$$\text{Accuracy} = 1 - \frac{\text{\#misclassified}}{\text{N}}$$

Evaluation

Metrics

- Recall, Precision and F_1 score for each malware family

$$\mathrm{Recall} = \mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\mathrm{Precision} = 1 - \mathrm{FDR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

$$F_1 = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

Result

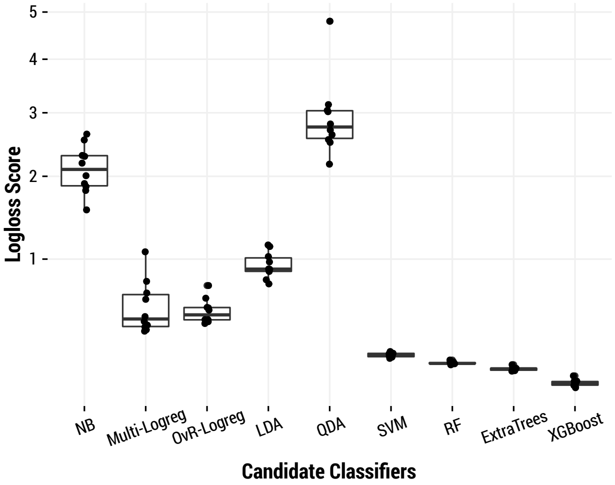


Figure 10: Model Evaluation: Multi-Class Logloss

Result

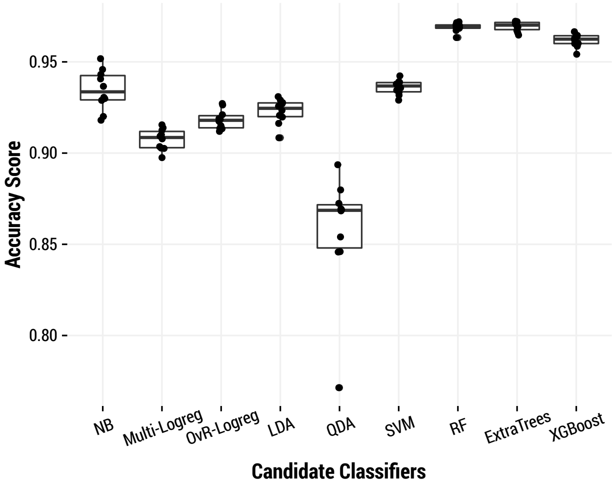


Figure 11: Model Evaluation: Multi-Class Accuracy

Result

Table 2: Confusion Matrix for Extremely Randomized Trees

1422	28	0	0	0	0	14	0
24	2329	0	0	1	0	0	0
0	9	2785	0	1	0	0	0
3	24	0	414	2	0	6	2
7	32	0	2	671	0	1	0
4	8	0	0	0	364	2	0
80	16	0	2	4	0	1065	0
2	6	0	0	2	0	12	940

Result

Table 3: Classification Report for Extremely Randomized Trees

Family	Rrecision	Recall	F_1 Score	Support
Ramnit	0.92	0.97	0.94	1464
Lollipop	0.96	0.99	0.97	2354
Kelihos_ver3	1.00	1.00	1.00	2795
Vundo	0.99	0.92	0.95	451
Tracur	0.99	0.94	0.96	713
Kelihos_ver1	1.00	0.96	0.98	378
Obfuscator.ACY	0.96	0.91	0.93	1167
Gatak	1.00	0.98	0.99	962
Avg / Total	0.97	0.97	0.97	10284

Question?

Thank You!