International Journal of Security and Its Applications
Vol. 9, No. 3 (2015), pp. 293-318
http://dx.doi.org/10.14257/ijsia.2015.9.3.23

On the Comparison of Malware Detection Methods Using Data Mining with Two Feature Sets

Sathaporn Srakaew Warot Piyanuntcharatsr and Suchitra Adulkasem

Dept. of Computing, Faculty of Science, Silpakorn University, Thailand
*Chantana Chantrapornchai**
Dept. of Computer Engineering, Faculty of Engineering,
Kasetsart University, Bangkok, Thailand
E-mail: fengcnc@ku.ac.th

Abstract

In this work, we compare the research methodology and performance of malware detection using data mining. Feature selection is an important problem in data mining. For the malware application, it is interesting to see which features that can be used to characterize the malware. Particularly, we are interested to compare two approaches that use features based on statistical values and the instructions. We adapt the experiment methodology using statistical features in [1] using 1,2,3 grams and varying block sizes as well as the methodology using the abstract assembly in [2] using 1,2,3 grams of consecutive instructions. We apply to our selected test set which is the data set from [3]. The decision tree J48 is used to model to detect three classes: Allapple, Podnuha, Virut. From the comparison experiments, it is found that the approach that considers the instruction set feature performs better. The test with the application set can give up to 100% correctness using the instruction features.

Keywords: *Malware detection, data mining, assembly features, statistical features*

1. Introduction

Malware is a malicious code or software which can be vulnerable to the system in many aspects. For example, it may turn off the security system of the computer and let the attacker find ways to destroy the system in some manners. It may help the attacker captures personal information like the credit card number, the bank account number, the login name or the password etc. It may bother the users by creating some unwanted processes which can slow down the system drastically, and more.

* Corresponding Author

ISSN: 1738-9976 IJSIA

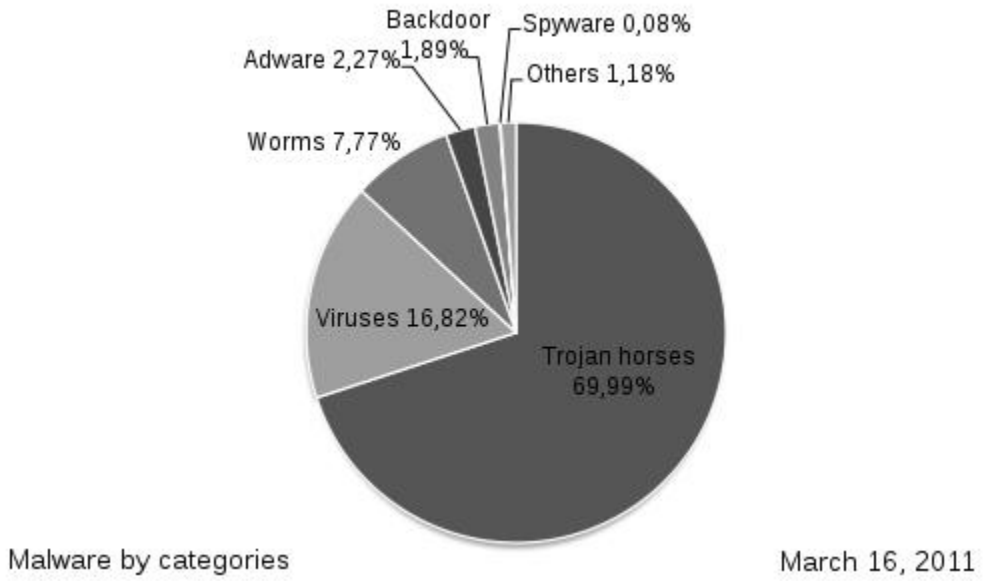


Figure 1. Malware by Categories (http://en.wikipedia.org/wiki/File:Malware_statics_2011-03-16-en.svg)

Software like firewalls, and anti-virus, or special hardware can help detect malware. Many are free and some comes with costs. For example, Malwarebytes (https://www.malwarebytes.org/free/) provides a free version and a premium version. In (http://www.techrepublic.com/blog/10-things/10-ways-to-detect-computer-malware/970/), it is mentioned that many tools to detect and remove malware for example, the anti-virus program like Avast or Comodo. SUPERAntiSpyware is a scanner for detecting and removing malware. Microsoft Baseline Security Analyzer is a program which detects the security vulnerability of the system setting and informs the users. Spychecker (http://www.spychecker.com/software/freeware_antispy.html) provides websites which list free malware detection tools for example, SpywareBlaster, Norton Power Eraser, RogueKiller, Combofix.

In this paper, we are interested to compare the method of data mining using different malware features. In particularly, the goal of the comparison is which kinds of features can characterize malwares better. We study two feature styles: the features that concern with information theory and the features related to instruction level affects the research methodology of data mining. We compare the performance of the method using statistical values of the binary and the abstract assembly of the malware binary. The research methodologies of both approaches for feature extraction are described.

2. Backgrounds and Related Works

Stalling classified malware into dependent and independent kinds [4]. The independent kind is the one that is executable by itself such as worms and bacteria. The dependent kind requires the host program to spread over such as Trojan horse, trapdoor, etc. In (http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html), the damage levels are varied. In anyway, malware cannot harm the hardware. Some commonly known types of malware are viruses, worms, Trojans, bots, back doors, spyware, and adware. However, the two common ones are worms and virus.

Let us look at some example of the malware types in http://pi1.informatik.uni-mannheim.de/malheur/#refset. Virus is an example of the dependent one. It is a program that contains the infection software. When it runs, it will spread to other executable files. Worm is a program that can send itself over the network to spread. Worm is a stand-alone one. Trojan horse is a form of program that attracts the user to run. It also needs a host file. When it runs, the malicious code will do bad things to the system such as deleting or ruining the user files, shutting down the system, turning off some security etc. Backdoor is an approach to bypass some authentication process. It is used before the attacker can enter the system. All apple is a multi-thread which is a polymorphic network worm. It will annoy the network service or server. Podnuha is kinds of virus which will slow down the system and let the hacker observe personal information. Virut is a kind of virus which can be spread with .exe and .scr. It will allow hackers to access the computer by opening the backdoor to the server.

One may categorize the malware detection method into static and dynamic kinds. The static approach considers the binary or the source code and detects the pattern. Mostly, the bytecode or hexcode string patterns are determined. The dynamic approach uses the runtime behavior. The behavior of the software at runtime is analyzed for detection.

Rossow, *et al.,* presented an interesting survey on many literatures for malware detection techniques [5]. Several pitfalls are made about the malware detection algorithms and experimental designs. The mentioned points are data set correctness, clarification issues in malware tested, platforms, samples, network etc. Further points are realism and safety. The realism is how realistic the approach is in the real-environments, how to evaluate the related families, what about the effect of the malware behavior on different OS, what stimuli are used for each malware. Muazzam, Morgan, and Joohan also proposed the survey on data mining techniques used in the malware detection based on the file features [6].

CWSandbox is a popular tool which can execute malware samples in a simulated environment. It was proposed by Willems, *et al.,* [3] the tool uses the API for interactions between malware and CWSandbox. The simulated environment can help observe the behavior of malware in a constrained manner.

To develop malware detection techniques, in [5], there are several points that need to be considered. First, it is the data set representation. Second, it is false negative that alerts we think they are. However, it is not true that the anti-virus may fail when it does not detect the virus. Thirdly, the false positive alerts when we may not see them. The anti-virus may fail even though it reports the virus either. The data set of malware is also important where Hosoi and Kanta proposed a common dataset for malware detection in the workshop of RAID 2010[7]. They have evaluated the dataset through observations and a questionnaire in their workshop.

It is noted that most of the previous work uses pattern matching of the string of hexadecimal values. Some comes up the tuple of positions and values while some uses the assembly instructions [8, 9]. Some even uses the text string of log files. Thus, the features are mostly come from results of the kind of matching found.

For example, Choudhary and Saharan also used data mining technique in malware detection [2]. They considered abstract assembly and selected top L features. IDApro was used to generate back the assembly code. SVM and Neural net classifiers are considered.

Huang, *et al.,* also presented an intelligence system for malware [10]. They designed the malware ontology and build the intelligent system on it.

In the network area, some used clustering to identify the malware. Kruegal, *et al.,* Considered Internet malware such as spam, worms, bots, spyware [11]. They clustered the malware based on the behavior. They considered three aspects: consistency, completeness and conciseness.

Perdisci, Leea, and Feamstera considered the network malware by looking at the behavior clustering of HTTP-based malware [12, 13]. The network behavior was studied, especially, the network traffic to generate the network signature. Seven statistical models for network behavior is considered: the number of HTTP requests, the number of GET requests, the number of POST requests, the average length of the URLs, the average number of parameters in the request, the average amount of data sent by POST requests and average response length.

FIRMA is also a clustering technique considering the network traffic behavior [14]. It deals with URL and HTTP. Based on the network signature generated, the malware is detected.

VAMO is an automatic malware clustering system where it considers the validity index [13]. Its goal is to solve the naming inconsistency of various anti-virus software.

Kumar and Mishra used the sequence alignment algorithm to detect malware [15]. It is based HEX-code and ASCII code transformation. The artificial neural network model is used. The tests for viral and worm were considered.

Tabish, Shafiq, and Farooq used the statistical approach to classify malware [1]. They consider three types of files:DOC, EXE, JPG, MP3, PDF, and ZIP. The 52 features are considered which include 1,2,3,4 grams of 13 statistical features. 6 types of malware are considered. Rieck, *et al.,* also used machine learning for malware detection. They used both clustering and classification. They proposed an incremental approach for behavior-based analysis. They collect the behavior of malware binaries on a daily basis. The data set is based on http://pi1.informatik.uni-mannheim.de/malheur/ under CWSandbox.

Pratheema, Prabha and Kavitha presented to use the naive bayes, KNN, and J48 to classify malware [8]. It also considered the HEX code with N-grams as features. 15 subfamilies were considered with the total of 1056 samples. Table 2 summarizes some of the previous works related to malware detection using data mining. They are differed in techniques, models, features, and data set used.

Table 2. Selected Previous Work in the Malware Detection Technique

Work	Data mining/ clustering	Data set	Network	Malware type	method
Pratheema, Prabha and Kavitha[8]	Data mining (classifier: NB,KNN,J48)	100 binary (training benign=90, malware=10)	-	15 subfamily	Hex code/n-grams of different size
Rafique and Calballero[14]	clustering	16,000 malware binaries	yes	Network signature	11 network feature
Bailey et.al. [11]	Classification	3,698	yes	Groups of malware	System state changes vector
Komashinskiy and Kotenko[9]	classification Decision Table C4.5 RandomForest Naive Bayes	Malware=5854 benign=1656	-	Malware and benign	<position, byte>
Choudhary and Saharan[2]	Classification : SVM, NN	200, 500 files	-	Type 1, Type 2	Instruction sequence
Kumar and Mishra[15]	Classification IBK	323 (virus+worm)	-	Virus, worm	Sequence alignment
Tabish, Shafiq and	Classification-	Benign=1800	-	backdoor,	Statistical

Farooq[1]	decision tree (J48)	and malware = 10, 311		trojan, virus, worm, construc-tor and miscellane ous	features

3. Methodology

In the following, we describe the research methodology for both approaches [1, 2].

3.1. Statistical Features Approach

This work experiments according to the statistical features proposed in [1]. The features computed are Simpson's index, Canberra distance, Minskowski distance, Manhantance distance, Chevbychev distance, Bray Curtts distance, angular separation, correlation coefficient, entropy, Kullback - Leibler Divergence, 3.2.12 Itakura-Saito Divergence, total variation[16].

Figure 2 presents the overall methodology and comparison scheme. We obtain the dataset from [3]. The data set contains the reference data set and application data set. The reference set is used for creating the model and the application set is used for testing the accuracy. Though there are many data classes, the malware classes selected are Allapple, Podnuha, Virut. The same methodology can be used for other classes.

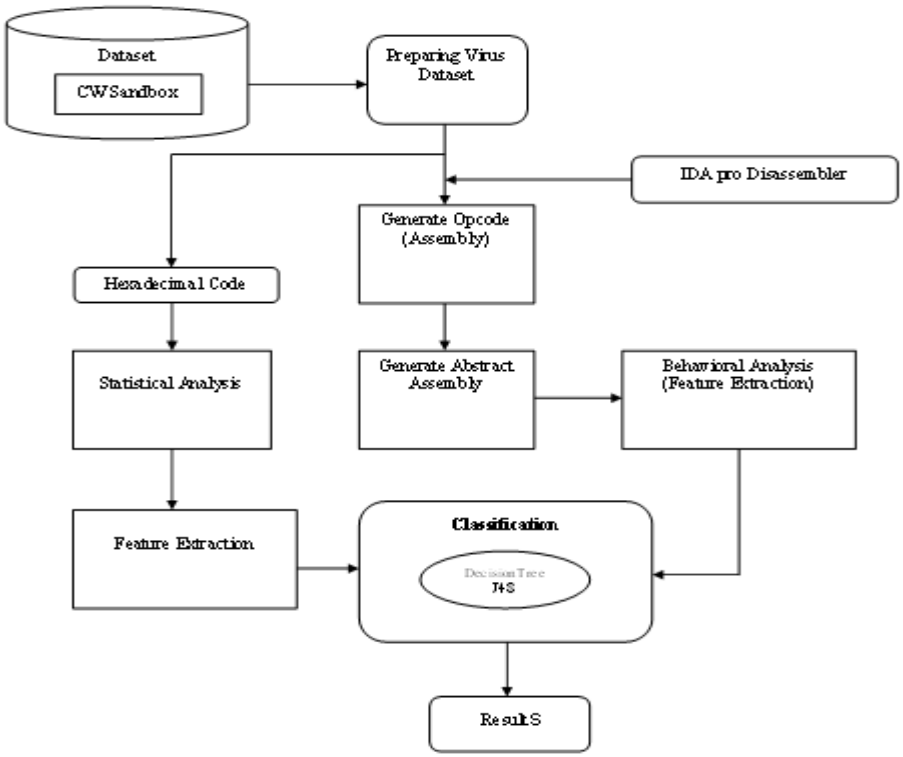


Figure 2. Testing Framework

For the statistical features, the malware file is transformed into the hexadecimal code where the statistical values are computed. The data set was obtained from CWSanbox available at http://pi1.informatik.uni-mannheim.de/malheur/. There are two classifications summarized in Tables 2-3.

1. Reference data set contains totally 1,200 files for training. This contains malware classes: Allapple, Podhuha, Virut, each for 300 files. Also, the normal files of types: DOC, JPG, EXE, each of them is 100 files.

2. Application data set contains 3,251 files. Each contains malware class Allapple, Podhuha, Virut, where each class has 890, 8, and 2,053 files respectively. The normal files for DOC, JPG and, EXE are 100 files each.

Then the files are converted into hexadecimal values for extracting statistical features using N-grams. For the instruction level, the files are converted to logical assembly using IDA Pro Disassembler (https://www.hex-rays.com/products/ida/). This is to generate opcodes to view consecutive instructions to extract features. Then, selected features are applied to WEKA (http://www.cs.waikato.ac.nz/ml/weka/) v.3.6.10 using decision tree J48 for classification since it outperforms other classifiers as experimented in [1].

Table 2. Reference Data Set

Types	Total files	Min Size (KB)	Max Size (KB)	AverageSize (KB)
ALLAPLE	300	57	837	71
PODNUHA	300	109	109	109
VIRUT	300	13	1,139	139
DOC	100	22	1,734	653
JPG	100	10	2,101	387
EXE	100	29	944	114

Table 3. Application Data Set

Types	Total files	Min Size (KB)	Max Size (KB)	AverageSize (KB)
ALLAPLE	890	56	176	70
PODNUHA	8	122	122	122
VIRUT	2,053	12	1,036	281
DOC	100	22	1,715	229
JPG	100	10	2,186	396
EXE	100	80	150	86

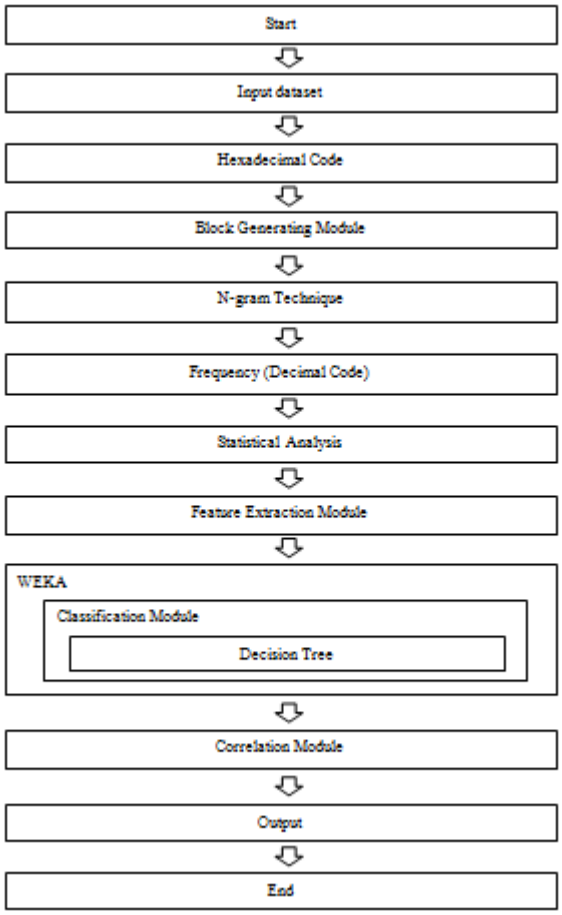


Figure 3. Methodology Steps for the Statistical Features Approach

Figure 3 presents the methodology steps in testing the approach that uses the statistical feature. The reference data set is transformed into the hexadecimal code (Figure 4). Then, the code is divided into blocks where we consider 50KB, 100KB, and the file size in Figure 5. Note that in [1], only 100KB is used. In our case, we also study the effect of block size as well for the statistical feature test. The file is converted to the decimal value to calculate the statistical features. Thirteen statistical values are computed and each will be considered as 1,2,3 grams. Totally, there are 39 features. The feature frequency is counted and used to model the decision tree in WEKA. After that, the WEKA classified the blocks. The blocks are correlated by the given threshold. For example, giving the threshold 0.5 means that the malware blocks must be more than 50%. Then, the file is considered malware. Otherwise, it is classified as Normal. The output results will report that the file is malware or not and it may be classified as which types.

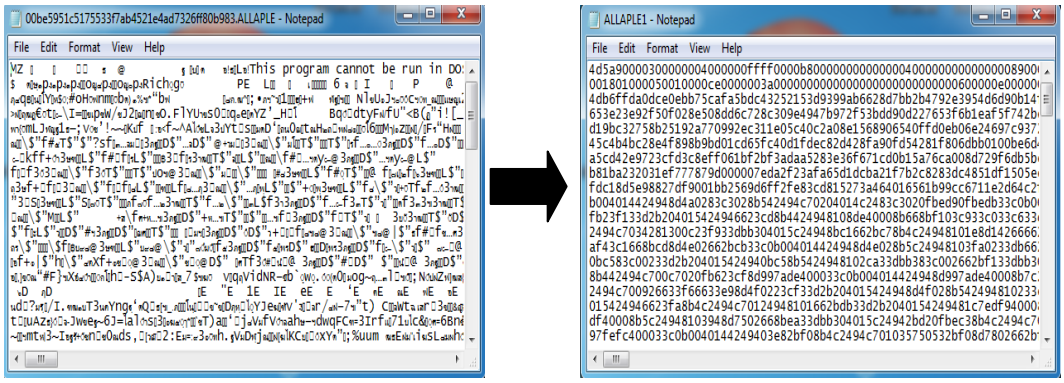


Figure 4. Hexadecimal Code

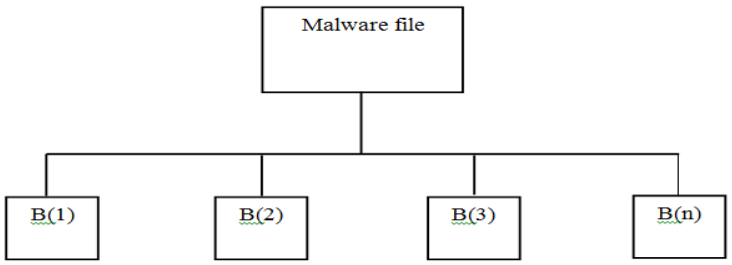


Figure 5. Block Division

For example, 4db6ffda0dce0ebb, we calculate 1,2,3 grams as:

1 gram : we obtain 4d | b6 | ff | da | 0d

2 gram : we obtain 4db6 | b6ff | ffda | da0d | 0dce

3 gram : we obtain 4db6ff | b6ffda | ffda0d | da0dce | 0dce0e

Then, the above hexadecimal values are converted to decimal values to count the frequency for each gram. Thus, the possible values for 1,2, and 3 grams are 0-255, 0-65,536, and 0-16,777,215 respectively. These decimal values are used to calculate statistical values (13 each, for 1,2,3 grams totally). The 39 feature values are saved as the ARFF file (Figure 6). Then, the decision tree is built.

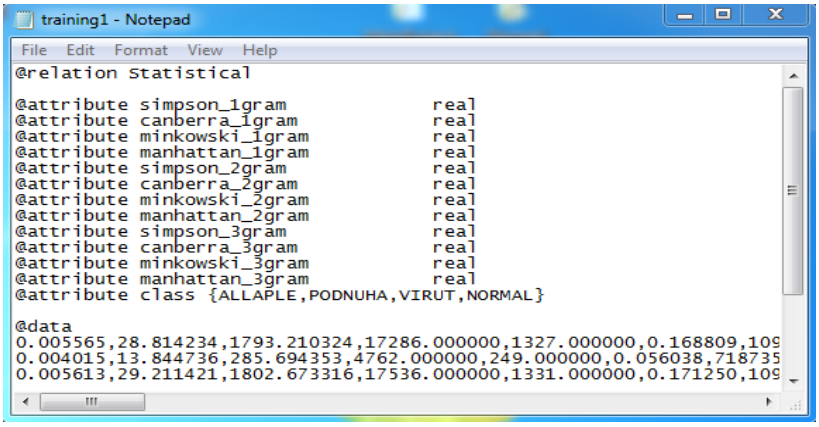
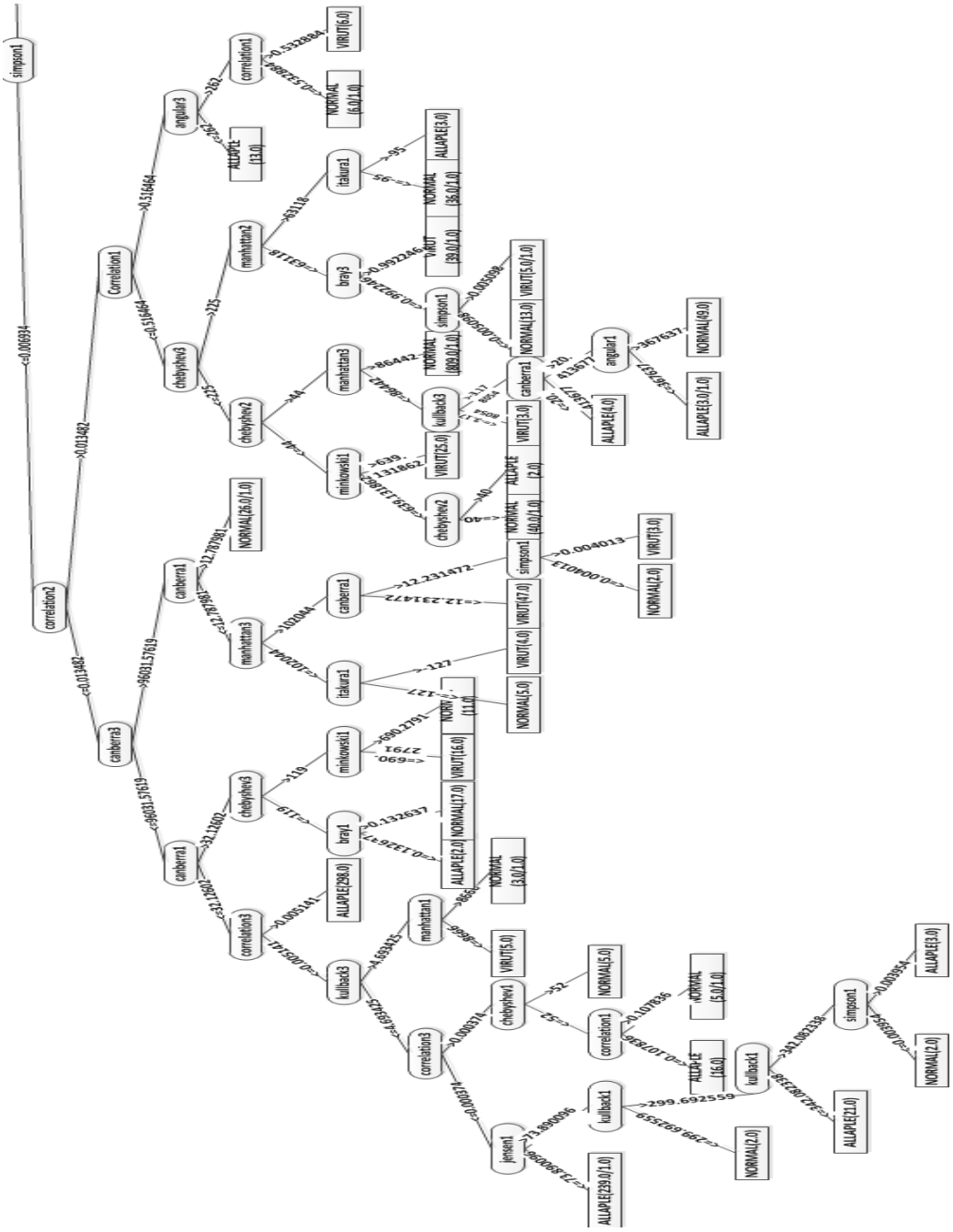
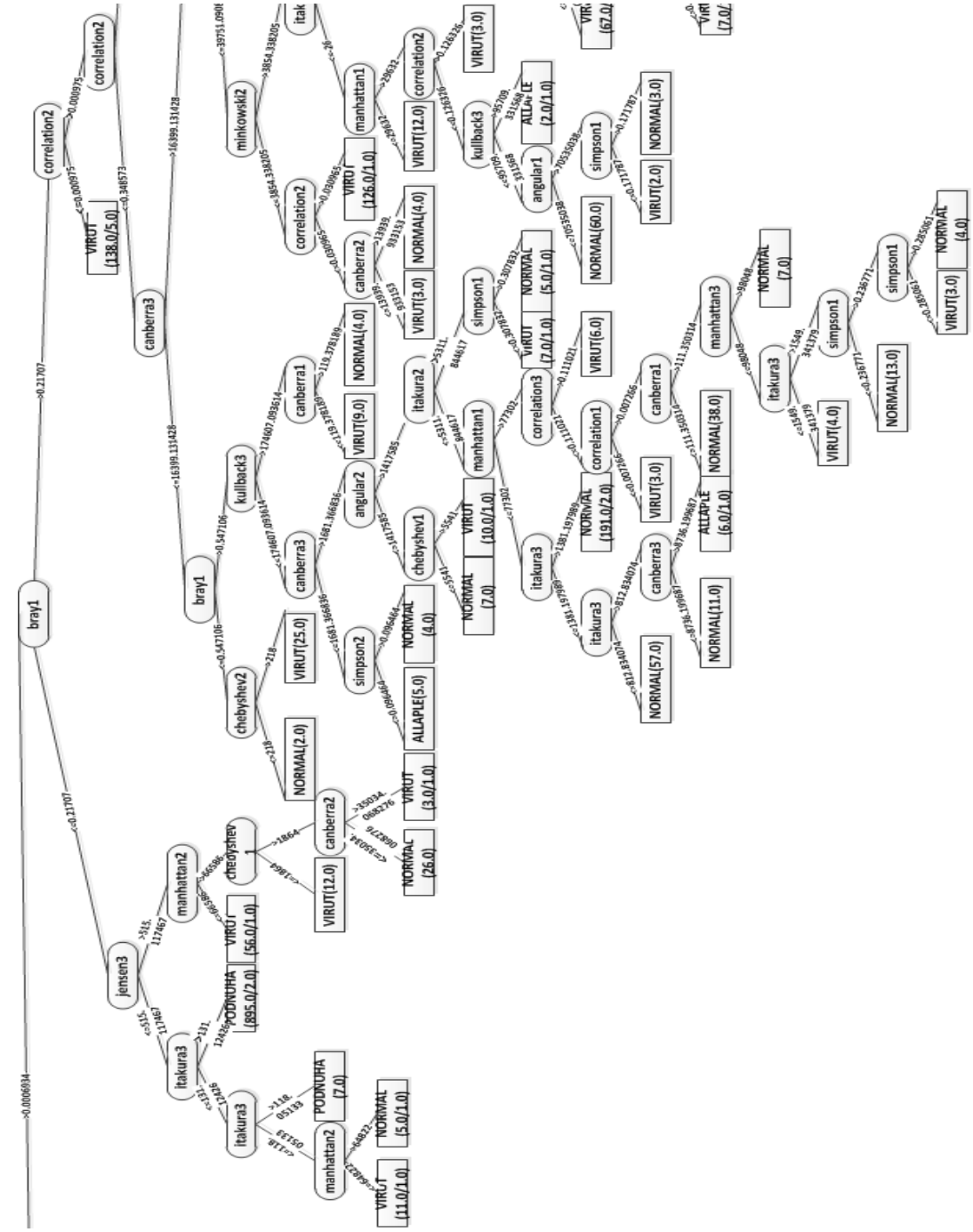


Figure 6. All Features in ARFF Format

Figures 7 present the tree for the case of 50KB block size and Figure 8 presents the tree for the case of block size = file size. It is noted that the latter tree is less complex. The number of nodes is smaller (39 nodes).





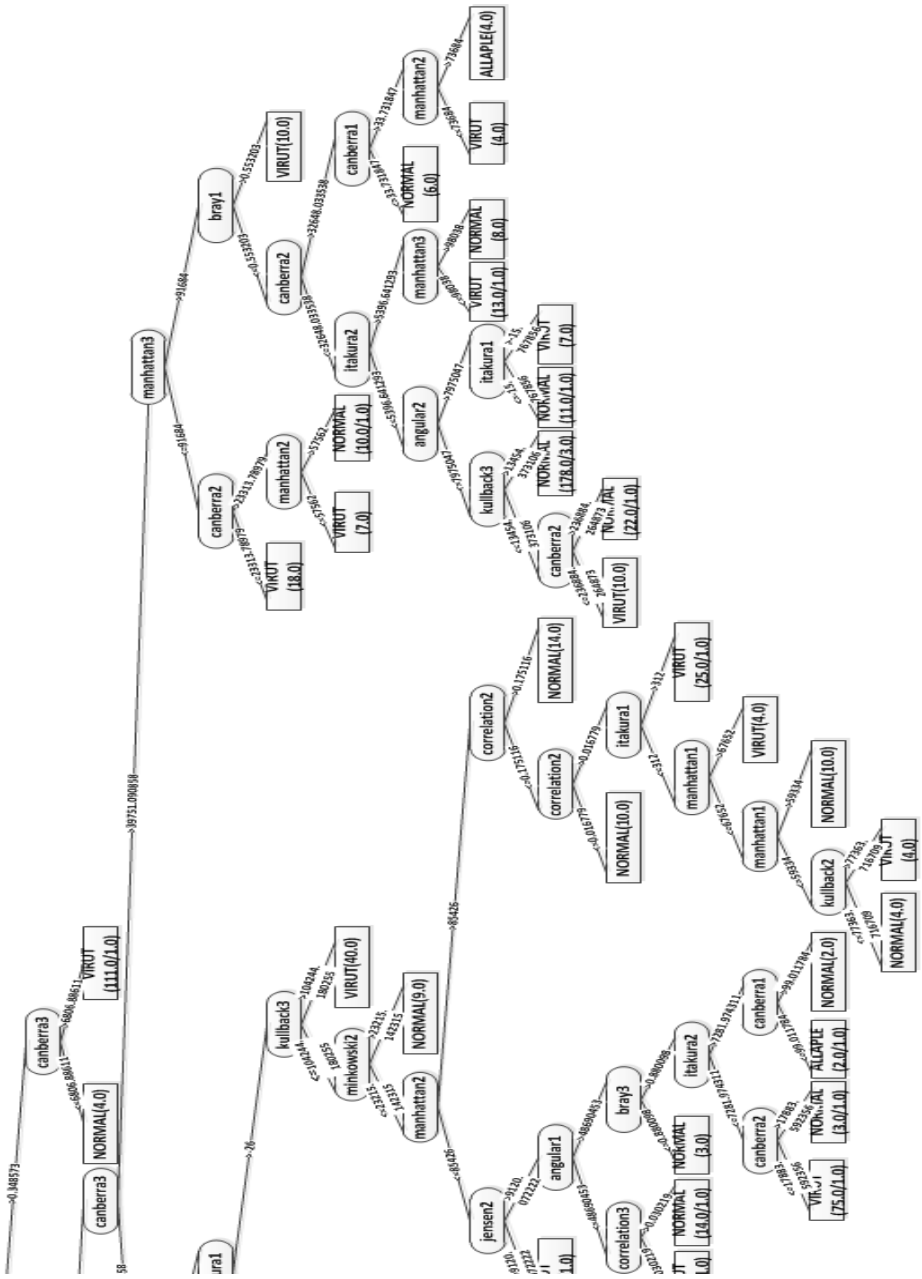


Figure 7. Decision Tree (J48) of Reference Data Set (Statistical Approach, Block Size=50KB)

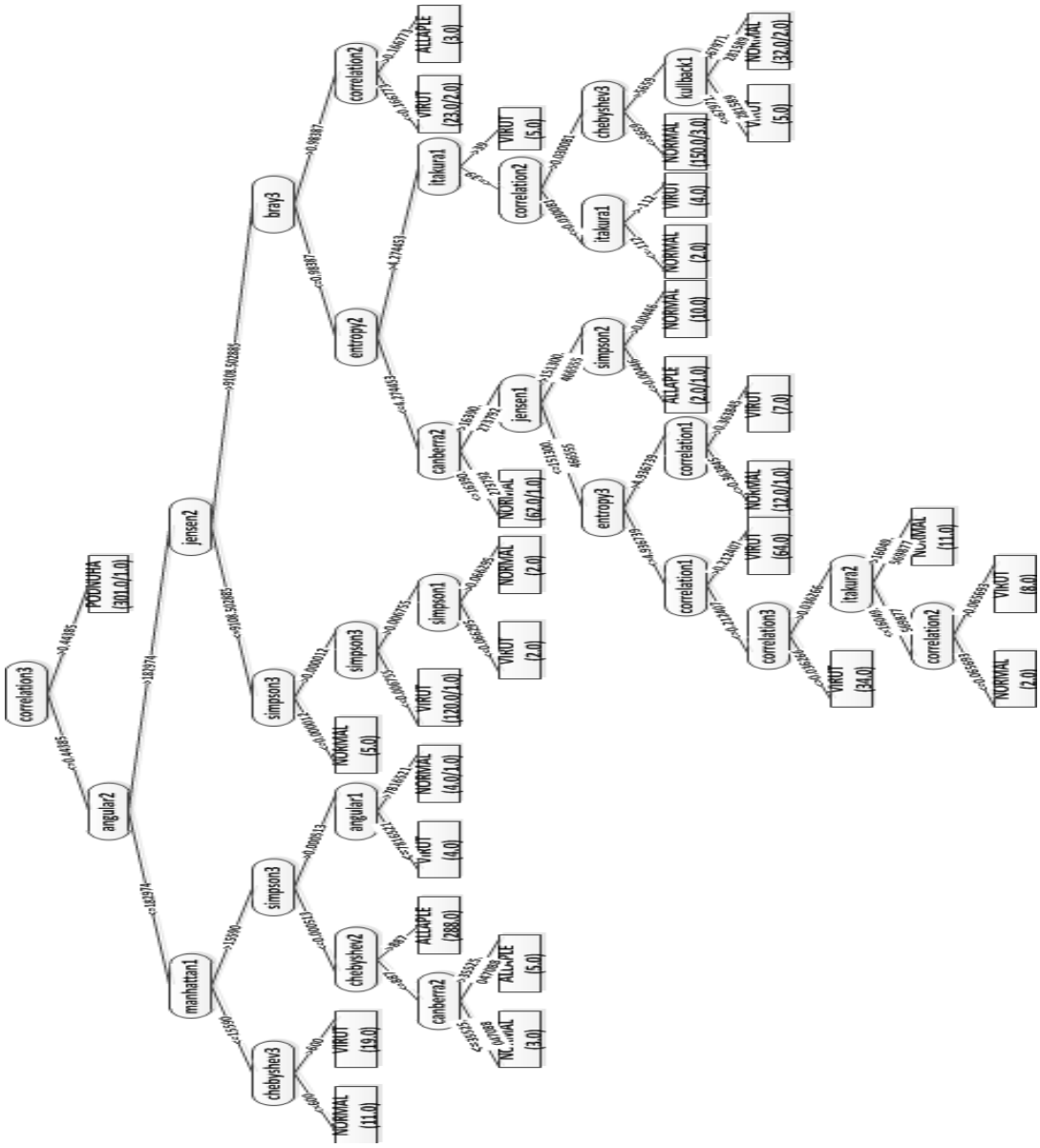


Figure 8. Decision Tree (J48) of Reference Set when Block Size = Original File Size

3.2 Abstract Assembly Approach

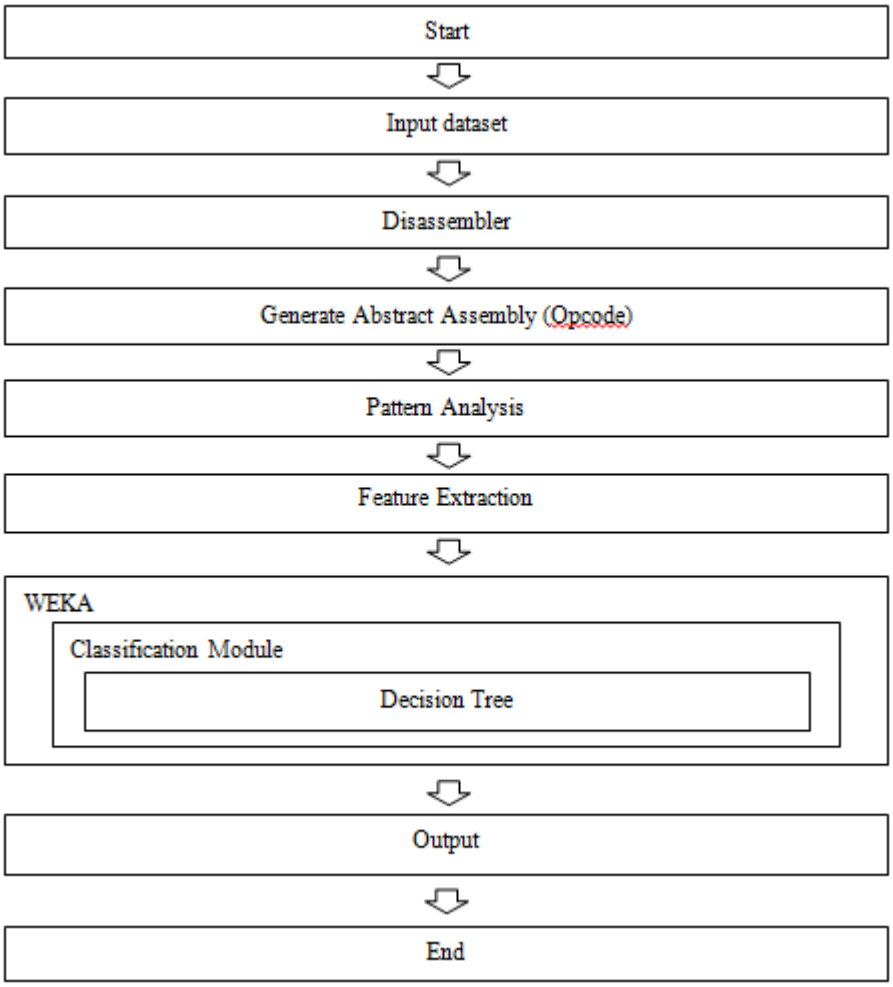


Figure 9. Method Steps for the Abstract Assembly Approach

In Figure 9, from the input file, IDApro is used to generate the abstract assembly (Figure 10). Here, we consider only the opcode field, where the operands are omitted. In the future work, types, number of operands and addressing modes of operands may also be considered. Then, the sequence of consecutive instructions are considered as 1,2,3 consecutive instructions and 1,2,3 grams.

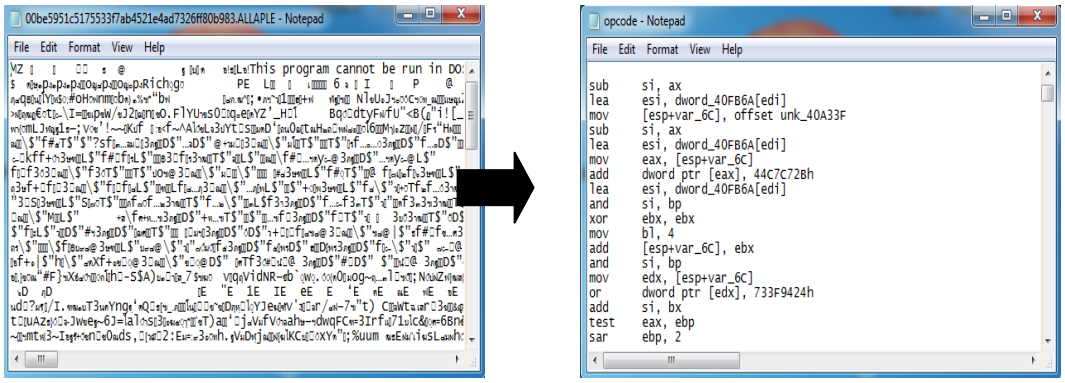


Figure 10. Transformation from the Binary to Abstract Assembly

Then we select 14 popular instructions [17] in Table 4.

Table 4. Frequent Malware Opcodes [17]

No.	Opcode
1	mov
2	push
3	call
4	pop
5	cmp
6	jz
7	jmp
8	lea
9	add
10	test
11	retn
12	jnz
13	xor
14	and

Then, we consider 1, 2, and 3 consecutive opcodes as in Figures 11-12.

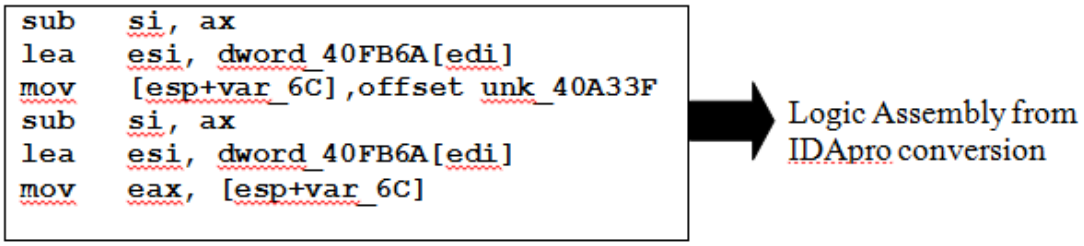


Figure 11. Logical Assembly Example

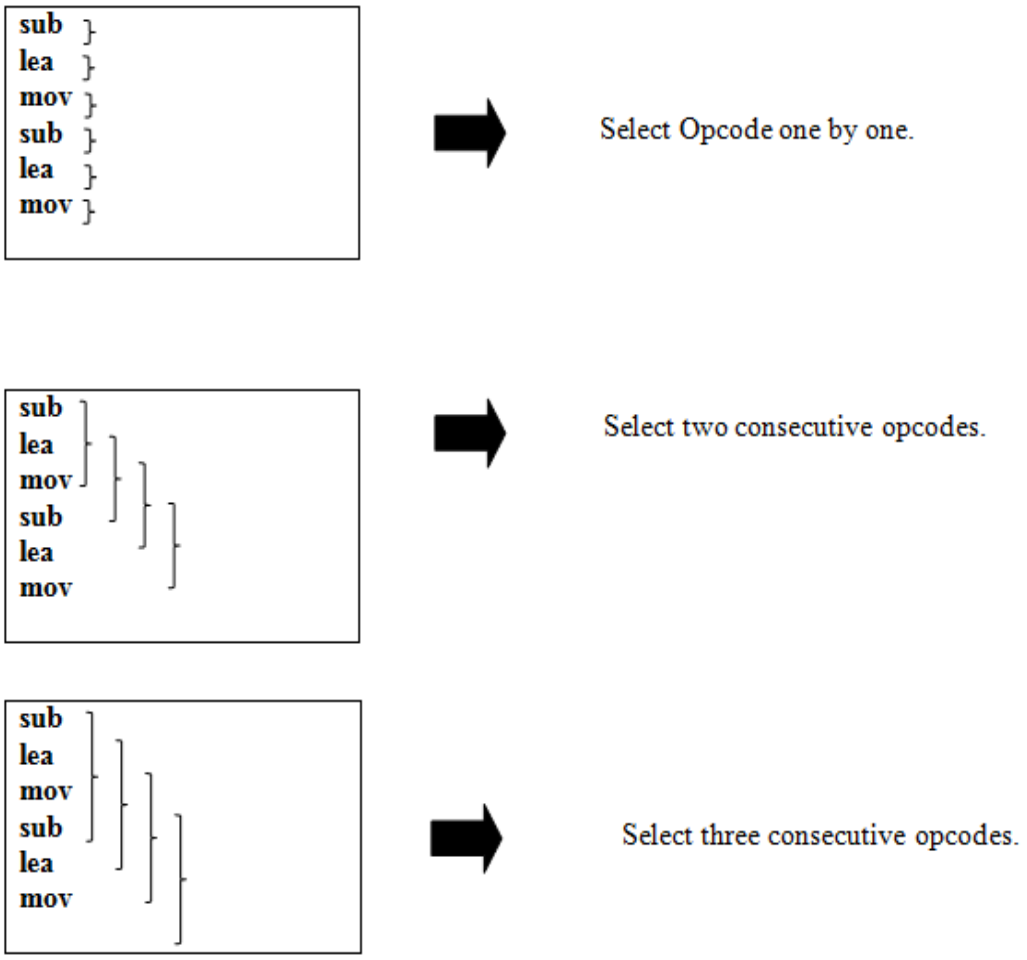


Figure 12. Opcode Selection for Three Cases

Then, we select top- frequency consecutive instructions as in Table 5.

Table 5. High Frequency of Consecutive Instructions

1 instruction	Freq.	2 consecutive instruction	Freq.	3 consecutive instruction	Freq.
mov	39.018088	mov|add	26.406250	xor|mov|add	22.157996
add	26.830319	xor|mov	18.958333	mov|add|add	4.110469
xor	18.432386	mov|mov	8.489583	mov|add|and	3.982017
and	5.727821	add|mov	4.739583	add|xor|mov	3.532434
lea	4.823428	add|xor	4.635417	mov|add|xor	3.468208
test	4.220500	add|add	3.593750	mov|add|lea	3.211304
push	0.473730	add|and	3.593750	lea|xor|mov	3.147078
pop	0.430663	add|lea	2.656250	mov|add|mov	3.082852
retn	0.043066	lea|xor	2.604167	mov|add|test	2.954399
call	0.000000	lea|mov	2.552083	and|xor|mov	2.376365
cmp	0.000000	add|test	2.552083	test|xor|mov	2.183687
jz	0.000000	mov|xor	2.239583	add|lea|mov	2.183687
jmp	0.000000	and|xor	2.187500	mov|xor|mov	2.119461
jnz	0.000000	and|mov	2.135417	add|mov|mov	2.119461

Totally, we obtain 58 attributes which are from 1 instruction for 14 attributes, from 2 consecutive instructions, for 17 attributes and from 3 consecutive instructions, for 26 attributes in the ARFF file (Figure 13).

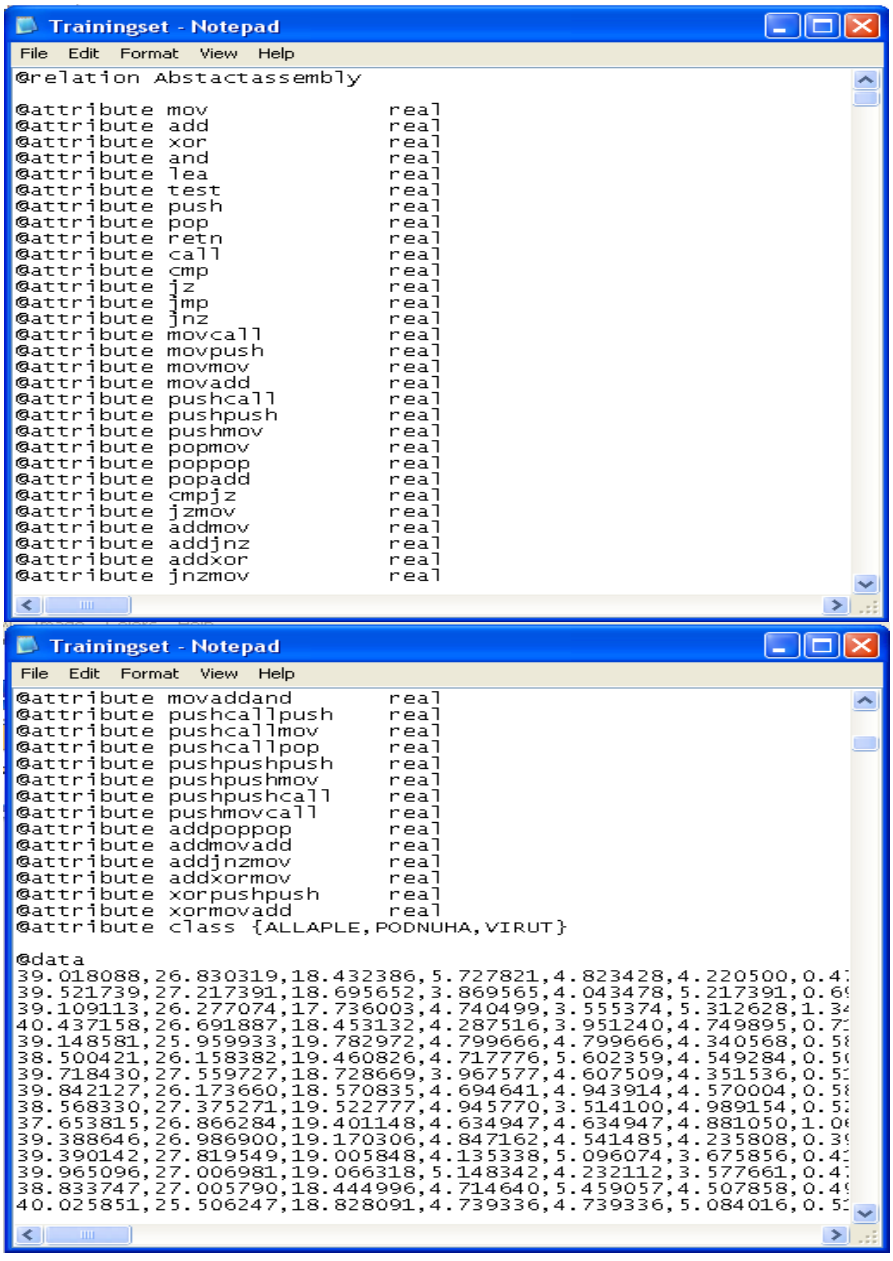


Figure 13. ARFF File for Abstract Assembly Features

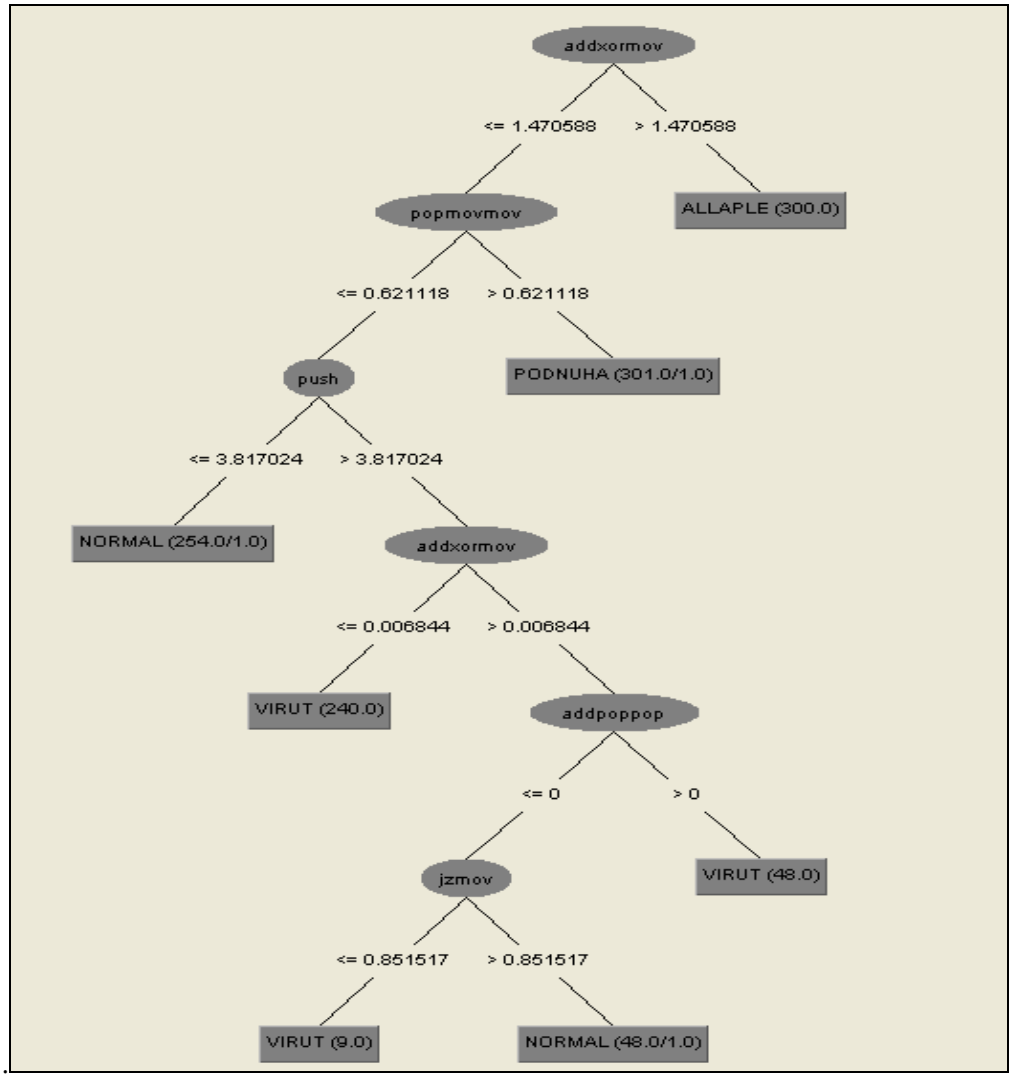


Figure 14. Decision Tree J48 Model (Reference Set) (Abstract Assembly)

Figure 14 presents the tree model of the reference set. It is seen that the size of tree is much smaller than the size of the tree obtained in Section 3.1. It is noted that in the future, we may consider other instruction-level features such as the addressing mode, the instruction length, instruction type, etc. Table 8 summarizes the methodologies of both.

Table 8. Comparison of Both Methodologies

Aspects	Statistical approach	Abstract assembly
Preprocessing	Convert to Hexadecimal codes.	Convert to assembly instructions.
Features	Statistical values	(Selected) Instructions
Grams	1,2,3 grams of bytes.	1,2,3 grams of interesting opcodes.
Block size	50K, 100K, File size	File size
Models	3 models since three cases of	One model since one

	block size.	block=file size.

4. Results

Let us characterize some results from the decision trees from each approach.

Table 9. Percentage of Correctly Classified and Incorrectly Classified Instances

Data set	Correctly classified (%)	Incorrectly classified (%)
Reference set	98.9167	1.0833
Application set	95.0477	4.9523
10-fold cross validation	95.333	4.667

Table 9 presents the correctly classified and incorrectly classified instances of the three cases of data set for 100KB-block case. From Table 9, it is seen that the application set has about 3% percentage less from the reference set percentage in the correctly classified instances and incorrectly classified instances.

From Table 10, the Podnuha has the highest TP. Normal has the high FP rate. Allapple has high precision in the application set. From all the tests, Virut is wrongly classified as Normal for the most number of instances (118 instances from the confusion matrix). Allapple has the highest precision values for the application set.

Table 10. TP, FP, Precision, Recall, F-Measure of the Statistical Methods

	TP	FP	Precision	Recall	F-measure
Reference set					
Allapple	0.99	0.001	0.997	0.99	0.993
Podnuha	**1**	0.001	0.997	**1**	0.998
Virut	0.973	0.003	**0.99**	0.973	0.996
Normal	0.993	**0.009**	0.974	0.993	0.997
Application set					
Allapple	0.978	0.001	**0.997**	0.987	0.987
Podnuha	**1**	0.001	0.667	1	0.8
Virut	0.94	0.025	0.985	0.94	0.962
Normal	0.943	**0.042**	0.695	0.943	0.801
10-fold cross validation					
Allapple	0.973	0.009	0.973	0.973	0.973
Podnuha	**1**	0.002	**0.993**	**1**	**0.997**
Virut	0.903	0.02	0.938	0.903	0.92
Normal	0.937	**0.031**	0.909	0.937	0.923

Table 11. Correctly Classified and Incorrectly Classified Instances (Abstract Assembly)

Data set	Correctly classified (%)	Incorrectly classified (%)
Reference set	99.75	0.25
Application set	98.39	1.661
10-fold cross validation	99.5	0.5

Table 11 shows the number of correctly classified and incorrectly classified instances of the different data set. From Table 11, the test on application set gives about 1% worse than that of the reference set and 10-fold cross validation which is better than the statistical features result.

Table 12. TP rate, FP rate, Precision, Recall, F-Measure (Abstract Assembly)

	TP	FP	Precision	Recall	F-measure
Reference set					
Allapple	**1**	0	**1**	**1**	**1**
Podnuha	**1**	0.001	0.997	**1**	0.998
Virut	0.99	0	**1**	0.99	0.995
Normal	**1**	**0.002**	0.993	**1**	0.997
Application set					
Allapple	0.998	0	0.998	0.999	0.999
Podnuha	**1**	**0.015**	0.14	**1**	0.246
Virut	0.975	0	**1**	0.975	0.987
Normal	**1**	0.002	0.984	**1**	0.992
10-fold cross validation					
Allapple	**1**	0.001	0.997	**1**	0.989
Podnuha	0.997	0.002	0.993	0.992	0.995
Virut	0.983	0	**1**	0.983	0.992
Normal	**1**	**0.003**	0.99	**1**	0.995

Table 12 shows that all the data set (reference set, application set, 10-fold cross validation) has a TP rate about 1. Podnuha and Normal have the TP rate 1 for the application set. The FP is also very small for all the data sets. Virut has the highest precision for the application set while in the statistical approach, Allapple has the highest precision value. From the confusion matrix, the Virut has the wrong instances as Normal only 5 instances and Virut is wrongly classified to Podnuha for 47 instances in the application set. However, compared to the statistical approach, the Virut is wrongly classified into Normal for 1,148 instances.

Next, we compare the correctness results from both approaches in both reference and application sets in details.

Table 12. The Correctness of Decision Tree J48 of the Reference Data Set

Block (KB)	Types	Total		Correctness	
		Total files	Total blocks	Total correct	(%)
50	ALLAPLE	300	633	618	97.6303%
	PODNUHA	300	900	900	100%
	VIRUT	300	984	969	98.4756%
	NORMAL (DOC, JPG, EXE)	300	1,789	1,774	99.1615%
	AVG				98.9549%
100	ALLAPLE	300	318	306	96.2264%
	PODNUHA	300	600	600	100%
	VIRUT	300	574	561	97.7352%
	NORMAL (DOC, JPG, EXE)	300	962	955	99.2723%
	AVG				98.6960%
SIZE	ALLAPLE	300	300	297	99%
	PODNUHA	300	300	300	100%
	VIRUT	300	300	292	97.3333%
	NORMAL (DOC, JPG, EXE)	300	300	298	99.3333%
	AVG				98.9167%

From Table 12, it is seen that when the block size is 50KB, for 1,200 samples, we can divide the files into "Total blocks". For instance, the ALLAPPLE files contain 633 blocks totally. Column "Total correct" displays the number of correct instances (by blocks) and column (%) is the percentage of correctness. It is seen that when the block size is large we obtain more percentage of correctness. The last case is when the block size is equal to the file size.

For the application data set, it is 95.04% for the maximum correctness in Table 13. Similarly, for Table 14, 10-fold cross validation gives the same results. The maximum correctness is when the block size equals to file size which is about 95.33%.

Table 13. The Correctness of Decision Tree J48 of the Application Data Set

Block (KB)	Types	Total		Correctness	
		Total files	Total blocks	Total correctness	(%)
50	ALLAPLE	890	1,801	1,635	90.7829%
	PODNUHA	8	24	24	100%
	VIRUT	2,053	6,894	6,314	91.5869%
	NORMAL (DOC, JPG, EXE)	300	1,682	1,590	94.5303%
	AVG				91.9431%
100	ALLAPLE	890	903	870	95.6044%
	PODNUHA	8	16	16	100%
	VIRUT	2,053	2,053	1,904	92.7423%
	NORMAL (DOC, JPG, EXE)	300	910	851	93.5165%

	AVG				93.7919%
SIZE	ALLAPLE PODNUHA VIRUT NORMAL (DOC, JPG, EXE)	890 8 2,053 300	890 8 2,053 300	870 8 1,929 283	97.7528% 100% 93.9601% 94.3333%
	AVG				95.0477%

Table 14. The Correctness of Decision Tree J48 of 10-cross Validation

Block (KB)	Types	Total		Correctness	
		Total files	Total blocks	Total correctness	(%)
50	ALLAPLE PODNUHA VIRUT NORMAL (DOC, JPG, EXE)	300 300 300 300	633 900 984 1,789	593 900 885 1,685	93.6809% 100% 89.9390% 94.1867%
	AVG				94.3567%
100	ALLAPLE PODNUHA VIRUT NORMAL (DOC, JPG, EXE)	300 300 300 300	318 600 574 962	295 599 506 900	92.7673% 99.8333% 88.1533% 93.5551%
	AVG				93.7245%
SIZE	ALLAPLE PODNUHA VIRUT NORMAL (DOC, JPG, EXE)	300 300 300 300	300 300 300 300	292 300 271 281	97.3333% 100% 90.3333% 93.6667%
	AVG				95.3333%

Table 15. Correctness of Decision Tree J48 (Reference Set) (Abstract Assembly)

Type	Total files	Correctness	
		Total correctness	(%)
ALLAPLE PODNUHA VIRUT NORMAL (DOC, JPG, EXE)	300 300 300 300	300 300 297 300	100% 100% 99% 100%
AVG			99.7500%

Table 15 shows the correctness of abstract assembly approach for the reference set. It is seen that the average correctness is high about 99.75%. Also, for the application set (Table 16), we obtain 98.33% and for 10-fold cross validation (Table 17), we obtain 99.5%. This is higher than the statistical approach.

Table 16. Correctness of Decision Tree J48 (Application Set) (Abstract Assembly)

Type	Total files	Correctness	
		Total correctness	(%)
ALLAPLE	890	888	99.7753%
PODNUHA	8	8	100%
VIRUT	2,053	2,001	97.4671%
NORMAL (DOC, JPG, EXE)	300	300	100%
AVG			98.3390%

Table 17. Correctness of Decision Tree J48 (10-fold Cross Validation) (Abstract Assembly)

Type	Total files	Correctness	
		Total correctness	(%)
ALLAPLE	300	300	100%
PODNUHA	300	299	96.6667%
VIRUT	300	295	98.3333%
NORMAL (DOC, JPG, EXE)	300	300	100%
AVG			99.5000%

Table 18 shows the comparison of the two approaches of the training set and Table 19 shows the comparison for the application set. It is seen that the abstract assembly approach performs better in both cases.

Table 18. Comparison of Correctness between Two Approaches of the Training Set

Approach	Total sample	Correctness	
		Total correct	(%)
Statistical	1,200	1,187	98.9167%
Abstract Assembly	1,200	1,197	99.7500%

Table 19. Comparison of Correctness between Two Approaches of the Application Set

Approach	Total sample	Correctness	
		Total correct	(%)
Statistical	3,251	3,090	95.0477%
Abstract Assembly	3,251	3,197	98.3390%

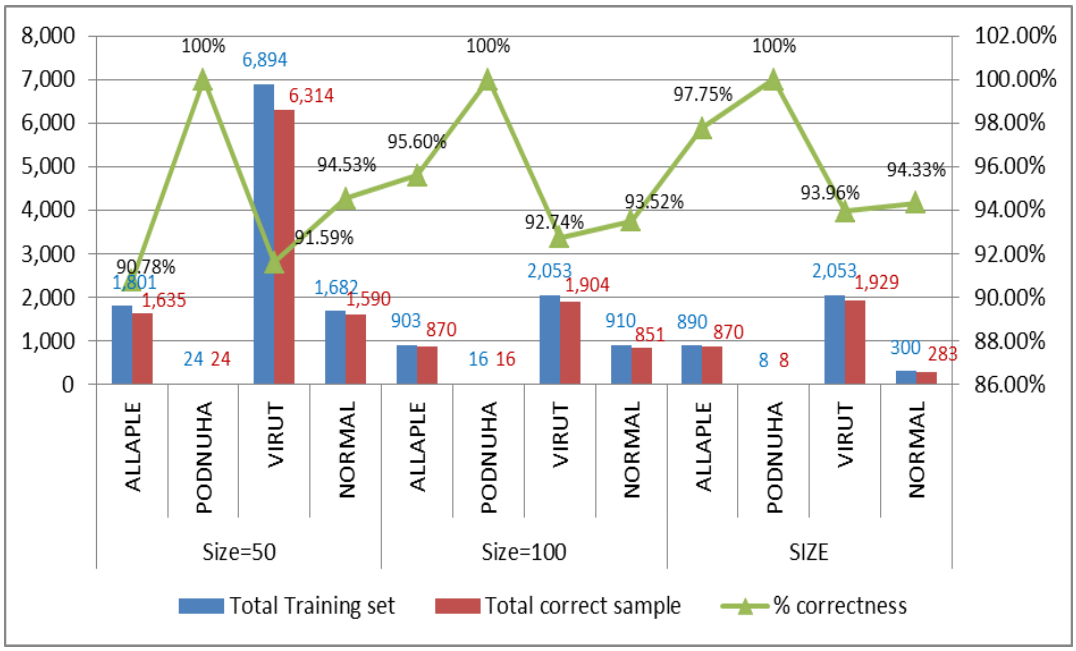


Figure 15. Comparison of Percent Correctness for each Malware Type Different Block size of Statistical Approach

Figure 15 shows the graphical results of the correctness by classes for the statistical approach. The method performs better when the block size is large. When the block size equals to the file size, it is best. Figure 16 shows the graphical results of the correctness by classes for the abstract assembly. The method performs well for all the cases.

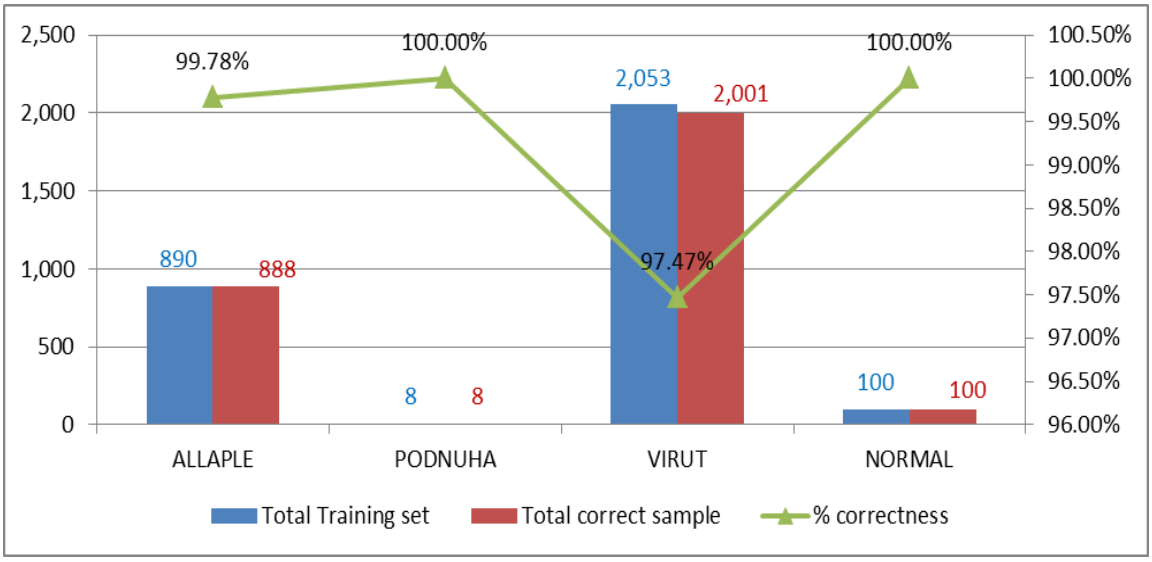


Figure 16. Comparison of Percent Correctness for Each Malware Class for Abstract Assembly Approach

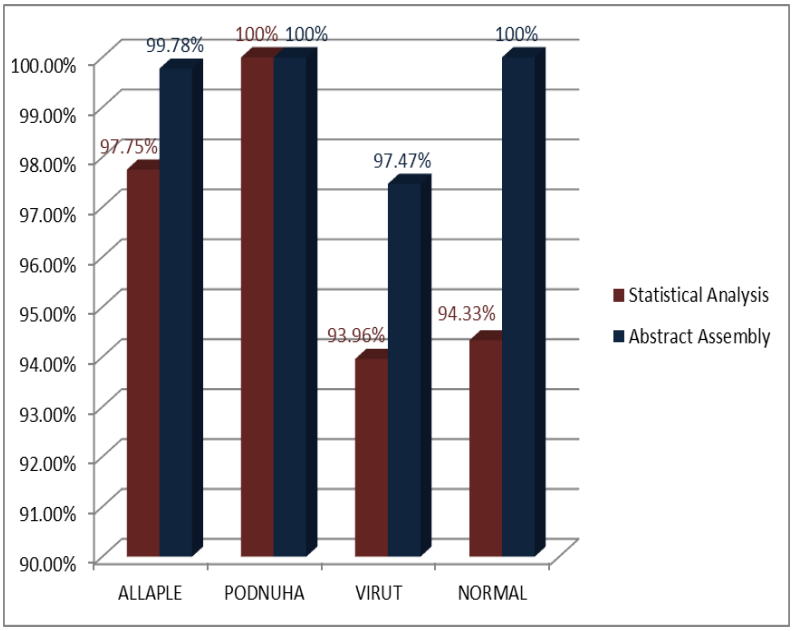


Figure 17. Comparison of Percent Correctness for Each Malware Class for Both Approaches

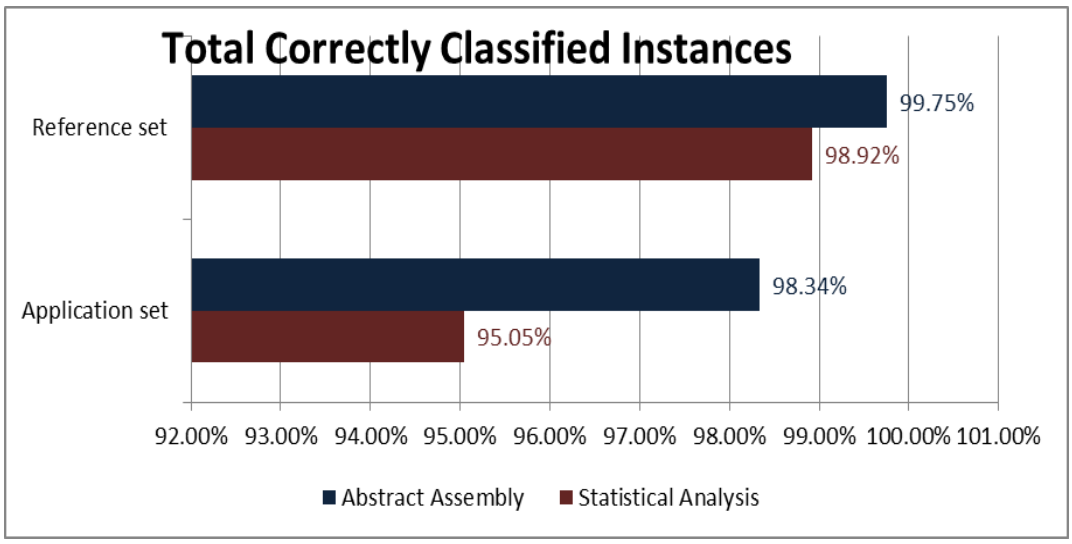


Figure 18. Comparison of Percent Correctness for Reference and Application Sets for Both Approaches

Figure 17 shows the comparison results of the correctness by class for the statistical method and the abstract assembly method. The abstract assembly method performs better about 2%-7% for all classes. On average, the abstract assembly method performs better for the reference set and the application set by 1-3% (Figure 18). Also, the precision, recall, F-measure is also better correspondingly (Figure 19). At last, Table 19 compares the performance of each method on the application set. It is seen that both performs well on Podnuha in general.

Table 19. Summary of the Performance of Each Approach

Items	Statistical method	Abstract assembly method
Number of correctly classified instances	95.04%	98.39%
Number of incorrectly classified instances	4.95%	1.661%
Highest TP rate	1 (Podnuha)	1 (Podnuha,Normal)
Highest FP rate	0.042 (Normal)	0.015 (Podnuha)
Highest Precision	0.997 (Allapple)	1 (Virut)
Highest Recall	1 (Podnuha)	1 (Podnuha,Normal)
Confusion matrix maximum error	118 instances (Virut classified as Normal)	47 (Virut classified as Pohnuha)

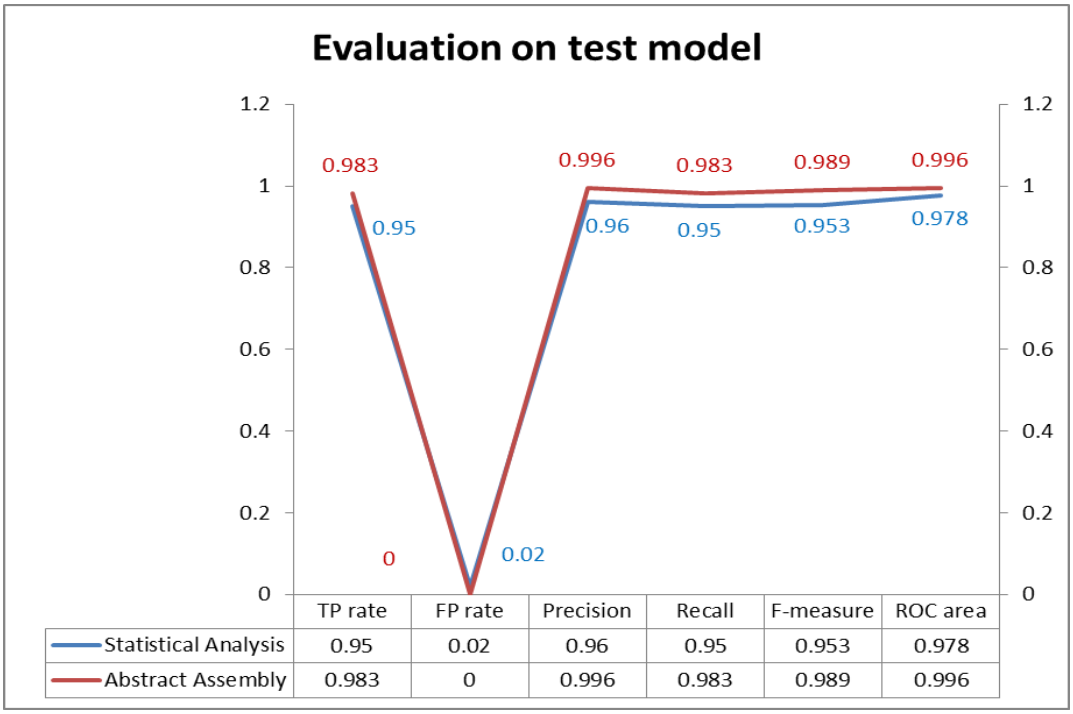


Figure 19. Efficiency of Both Approaches

5. Conclusion and Future Work

In this work, we compare the performance of two malware classification methods using data mining. Two methods use the different features: the statistical features and abstract assembly features. The decision tree is used to model as in the previous works.

For the statistical features, we use 13 statistical features and 1,2,3 grams to compute the feature values. The file is divided into blocks while computing these. The different block size is also considered. Thus, we obtain the model for each block size case. For the abstract assembly features, we map the code back to assembly language and count the interesting instructions. Methodology of collecting these features is presented.

The performance is measured by the correctness against the selected malware benchmark in [3]. It is observed that the abstract assembly approach is more promising, giving high accuracy with less complicated model, *i.e.*, the tree with 13 nodes while the using statistical feature leads to the tree of 39 nodes for the minimum size (the case of block size= file size).

Due to its good performance, and since we are interested in architecture level, more features can be observed, for example, opcode and operand, addressing mode, repeated block of code, branch addresses, *etc.*, Also, the combination with the dynamic style may be used. For example, the state of the execution (register values, memory values, *etc.*,) can be used to help detect some pattern. The dynamic instrumental tool such as pintools (http://software.intel.com/en-us/articles/a-dynamic-binary-instrumentation-tool) can be used to collect architecture behavior at runtime such as instruction mixes, instruction count, branch address, memory reference etc.

References

[1] S. M. Tabish, M. Z. Shafiq and F. Muddassar, "Malware detection using statistical analysis of byte-level file content", Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, ACM, **(2009)**, Paris, France.
[2] R. Choudhary and R. Saharan, "Malware detection using data mining techniques", International Journal of Information Technology and Knowledge Management, vol. 5, **(2012)**, 85-88.
[3] C. Willems, T. Holz and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox", Security & Privacy, IEEE, vol. 5, **(2007)**, pp. 32-39.
[4] W. Stalling, "Cryptography and Network Security", 4 ed. Pearson Education, **(2005)**.
[5] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos and M. V. Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook", Proceedings of IEEE Symposium in Security and Privacy (SP), **(2012)**, pp. 65-79.
[6] M. Siddiqui, M. C. Wang and J. Lee, "A survey of data mining techniques for malware detection using file features", Proceedings of the 46th Annual Southeast Regional Conference on XX (ACM-SE46), **(2008)**, Auburn, Alabama.
[7] H. Takurou and K. Matsuura, "Evaluation of the common dataset used in anti-malware engineering workshop 2009", Proceedings of the 13th international conference on Recent advances in intrusion detection, Ottawa, Ontario, Canada: Springer-Verlag, **(2009)**, pp. 496-497.
[8] A. Pratheema, M. Prabha and P. Kavitha, "Malware Classification through HEX Conversion and Mining", Proceedings of International Conferenece on E-Governance & Cloud Computing Sevices(EGov '12), **(2012)**, pp. 6-12.
[9] D. Komashinskiy and I. Kotenko, "Malware Detection by Data Mining Techniques Based on Positionally Dependent Features", Proceedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), **(2010)**, pp. 617-623.
[10] H.-D. Huang, T.-Y. Chuang, Y.-L. Tsai and C.-S. Lee, "Ontology-based intelligent system for malware behavioral analysis", Proceedings of IEEE International Conference on in Fuzzy Systems (FUZZ), **(2010)**, pp. 1-6.
[11] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian and J. Nazario, "Automated Classification and Analysis of Internet Malware", Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, vol. 4637, **(2007)**, pp. 178-197.
[12] R. Perdisci, W. Lee and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces", Proceedings of the 7th USENIX conference on Networked systems design and implementation San Jose, California: USENIX Association.
[13] R. Perdisci and M. U. Vamo, "Towards a fully automated malware clustering validity analysis", Proceedings of the 28th Annual Computer Security Applications Conference Orlando, Florida: ACM.
[14] M. Z. Rafique and J. Caballero, "FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors", Research in Attacks, Intrusions, and Defenses. vol. 8145: Springer Berlin Heidelberg, vol. 8145, **(2013)**, pp. 144-163.
[15] V. Kumar and S. K. Mishra, "Detection of Malware by using Sequence Alignment Strategy and Data Mining Techniques", International Journal of Computer Applications, vol. 61, **(2013)**, pp. 16-19.
[16] T. M. Cover and J. A. Thomas, "Elements of Information Theory", Wiley-Interscience, **(1991)**.
[17] B. Daniel, "Opcodes as predictor for malware", Int. J. Electron. Secur. Digit. Forensic, vol. 1, **(2007)**, pp. 156-168.

