

# **Development of an algorithm for malware detection using General Purpose Graphics Processing Unit**

Research Practice

Submitted for partial fulfillment of the requirement of BITS G540  
Research Practice

By

**Mayank Chaudhari**

2016H103014G

Under supervision of

**Dr. S. K Sahay**

Assistant professor

Department of Computer Science and Information Systems

Research Practice



**BITS Pilani**  
K. K. Birla Goa Campus



**MAY 2017**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,  
PILANI, K. K. BIRLA GOA CAMPUS**

## **CERTIFICATE**

This is to certify that the Research Practice entitled, Development **of an algorithm for malware detection using General Purpose Graphics Processing Unit** is submitted by **Mayank Chaudhari**, ID No. **2016H103014G**, for the partial fulfillment of the work done by him under my supervision.

Date:

(Signature of Mentor)

Place:

Dr. S K Sahay

Assistant professor

Department of CS&IS

BITS-Pilani, K. K Birla Goa campus

## **ACKNOWLEDGEMENTS**

I would like to thank my project Mentor Dr. Sanjay K Sahay for his valuable and constructive suggestion during the planning and development of this work. I am particularly grateful for the guidance given by Mr. Hemant Rathore and Mr. Ashu Sharma. The frequent reviews and suggestions provided by them were critical for the completion for this work.

## **ABSTRACT**

This project aims to develop an algorithm to classify Win32 executables as malware or benign using Naïve Bayes classifier on GPU. Being a static approach the executable files need not to run either on CPU, GPU or any other virtual environment during detection phase.

In this approach we try to group the executables into groups on the basis of their sizes during the training phase. The trained model is then used to detect the existing and new malwares. The use of Naïve Bayes algorithm makes it simple and easy to understand and implement. The use of GPU for general purpose computing and increasing number of malwares to process was the main motivation for this project.

# **CONTENTS**

I. Abstract	.... 2
II. Acknowledgement	.... 3
1. Introduction	.... 5
2. Malware types and detection techniques	.... 7
2.1. Introduction	.... 7
2.2. Malware classification	.... 8
2.3. Malware detection techniques	.... 9
3. Hardware Specification and CUDA programming	....11
3.1. Hardware specification	....11
3.2. CUDA programming model	....12
4. Previous work	....13
5. Naïve Bayes	....14
5.1. Introduction	....14
5.2. Feature selection	....15
5.3. Grouping	....15
6. Current work	....16
6.1. Implementation	....16
6.2. Training module	....17
6.3. Testing module	....18
6.4. Current status	....18
7. Conclusion and results	....20
7.1. Results and Future work	....20
7.2. Conclusion	....21
III. Bibliography	....22
IV. Appendix	....23

# CHAPTER 1

## INTRODUCTION

The widespread use of information technology and reliance on computers has made the information a valuable asset. Computer malwares are a major threat to today's networked environment. Malwares can be designed to steal, destroy or modify the information. The effect is leakage of sensitive information which can further be used for other malicious tasks affecting human lives in various forms. The most commonly used method for malware detection is signature based detection, that uses specific features of an executable to classify it as malware or benign. In our approach we use the opcode frequency of executables as signature for creating our model. This approach is not effective for previously unknown/self-modifying malwares, due to the same reason there is need to work on a more generalized approach for malware detection. The approaches to tackle this problem can be put under static malware analysis and dynamic malware analysis. Static analysis analyzes the structural properties of the malwares without executing the program under inspection (PUI), on the other hand dynamic analysis monitors the behavior of PUI by running under a sandbox.

For decades' researchers have been working to find more efficient techniques for detection of malware. With the advancement of technology many new techniques such as data-mining and machine learning are also being used in addition to the existing traditional techniques. The drawback of these techniques is that they are compute intensive and slow so this work is focused on developing a fast and efficient solution for malware detection. In this project static malware detection approach is used for detection of

malwares. by using Naïve Bayes classification technique. This approach can be divided into two parts namely training and testing. The other challenge we are dealing in this project is to make the detection process as fast as possible by using GPU for malware detection. The aim is to make things as much parallel as possible. This is required because in server class machines where multiple requests are coming simultaneously and we can't wait for much longer time for responding the requests. This work is an extension of work done under my mentor Dr. S. K Shay on static malware detection.

## **Report Layout**

Chapter 2, discuss types of Malware and different detection techniques. Chapter 3 discuss the Hardware specification and CUDA programming model for parallel implementation of our work on General Purpose Graphic Processing Unit. Chapter 4 discuss the related previous work. Chapter 5 discuss the basics of Naïve Bayes. Chapter 6 discuss implementation of our model. Chapter 7 discuss the results and conclusion.

## CHAPTER 2

# MALWARE TYPES AND DETECTION TECHNIQUES

### 2.1 Introduction

Programs can be classified as *good-ware/benign* or *malwares*. Good-wares are useful programs and do not cause any harm to the user or to the system. While on other hand Malwares are programs written with a malicious intent to harm a system, network or user. Malware is generalized term used for malicious program families which contain viruses, worms, trojans, spywares, adware's, rootkits, botnets, bacteria, droppers, key loggers, ransomwares etc.

*Viruses* are malicious programs that generally do not exist independently, they require a host program for propagation. Viruses infect files by injecting code inside programs which are used for further infection. *Worms* use network services for propagation and infection, unlike viruses they do not require any specific application or program. *Trojans* are programs that give a delusion of being a good ware but perform malicious once they execute. *Spywares* are used spying and stealing information from user's system and sending it to the specific server. *Adware's* are not malicious in actual sense but are used to bombard user system with various advertisements. *Rootkits* are used to gain system level access and are sometimes used in combination with worms and other malwares to enable hiding features. *Botnets* consists of a network of infected hosts which receives instructions from bot herder and perform malicious attacks or operations in a distributed manner. *Bacteria* are programs used for eating



up your memory. *Ransomware* are new type of malwares which ask to perform some action in order to give back the control of your system back to you after infection.

## **2.2 Malware classification**

Malwares can be basically divided into two generations based on the general structural properties. First generation has a fixed internal structure while second generation malwares change their structure in every variant or execution. Second generation malwares can be further categorized as follows.

### **Encrypted Malwares**

These type of malwares consist of two parts

- 1) Encrypted malware body that is used for malicious work during execution
- 2) A decrypting routine used for decrypting the malware body.

The decrypting routine signature can be used for detection of such viruses.

### **Oligomorphic Malwares**

Oligomorphic malwares are also similar to Encrypted malwares in a sense that they also use encryption techniques with a property of slightly

changing the decryptor routine with every execution. They can produce a few hundreds of variants.

### **Polymorphic Malwares**

Polymorphic malwares can create millions of variants of itself by changing instructions and are immune to signature based detection. They include a mutation engine that which on every execution generate a new decryption routine. These variants use obfuscation technique to achieve this property. Some common obfuscation techniques are dead-code insertion, code transformation, code transposition, register reordering, opaque predicates. Signature scanning can be used for detection of such malwares.

### **Metamorphic Malwares**

These are second generation Malwares and unlike their predecessors they change the execution code instead of changing only the decryption routine. Due to this reason detection of this category is a challenging task. They also use obfuscation techniques to deceive detection.

## **2.3 Malware Detection**

The type of risk and damage intended by a malware can be known after its detailed behavior. The malware analysis can be static and dynamic based on the techniques used for analysis.

## **Static Analysis**

Static malware analysis deals with the structural analysis of malwares. During this process malware is not executed and only malware code is analyzed in order to know the actions performed by it. The malware code is decompiled which gives insight of the malware code. Some of the techniques used in this process are byte stream signatures, opcode frequency analysis, control flow graph analysis, hash signatures, cryptographic hashes, fuzzy hashes etc.

Obfuscation techniques like code transposition, code movement, code substitution confuses the analyzers. The static analysis techniques alone are not enough to detect the malwares so there is a need of dynamic analysis of the malwares.

## **Dynamic Analysis**

Dynamic analysis checks the malware behavior by running the malwares either in a virtualized environment or on a real system for analyzing the execution behavior of the malware. Dynamic behavior monitors the network patterns, memory access patterns, process behavior and many other system dynamics to analyze the malwares. This process is resource and computation intensive.

Use of sandboxes and virtualized environments may sometime fail to analyze the virus behavior due to the techniques used by malware writers to evade such type of detection environment. Dynamic analysis is able to identify the zero day attacks also.

## **CHAPTER 3**

# **HARDWARE SPECIFICATION AND CUDA PROGRAMMING MODEL**

### **3.1 HARDWARE SPECIFICATION**

The project consists of two implementations of the project serial and parallel. The two setups are needed to show the benefit and performance gain obtained by the GPU over the sequential algorithm. The performance gain obtained in this work due to the use of GPU enables us to process more files simultaneously and encourages us to use GPUs in other implementations also to deploy faster detection techniques.

We are running the algorithm on intel Pentium processors with 3.5 Ghz clock speed and 2 GB RAM. For Parallel implementation we are use Nvidia Quadro K2000, Kepler series based graphic processor unit. The GPU consist of 384 CUDA cores divided into set of 2 streaming processors of 192 cores. We have 64KB on chip configurable memory which is divided into shared memory and L1 cache. This memory is equivalent to CPU cache memory and can be programmed as per the programmers use. The available configurations for this series are 32 KB shared and 32 KB L1 cache, 48 KB shared and 16 KB L1 cache, 16 KB shared and 48 KB L1cache. In addition to it the GPU has a DRAM of 2GB.

The one disadvantage of Kepler based architecture is that the L1 cache cannot be used for caching of Global reads from DRAM. We are using CUDA toolkit 7.5 for programming support and CUDA profiler for application profiling and performance monitoring.

## 3.2 CUDA programming model

CUDA is a parallel programming platform and programming model invented by NVIDIA. This programming model helps in exploiting the massive compute power of underlying CUDA based GPGPU in a system. This project uses *CUDA C API* for parallel implementation of the algorithm.

### CUDA C

CUDA C is a subset of C with some extensions. CUDA C provides us features to exploit data level parallelism by using GPU kernels. The Kernel is a function which is responsible to complete the job. Each thread in the system executes same kernel simultaneously, this is similar to the multithreaded application's in our normal system but at a large amount due to the presence of hundred time more cores present in a GPU.

The program sequence starts from writing a serial main function as our normal multithreaded program then memory is allocated and data is copied to GPU DRAM using *cudaMalloc ()* and *cudaMemcpy ()* functions. After copying the data to GPU we invoke the kernel function with required no of threads which are denoted by a 1 dimensional structure but can also be represented in a 2 dimensional representation. On completion of execution the data is copied back to host memory by memory copy functions.

## **CHAPTER 4**

### **PREVIOUS WORK**

This work is an extension of some previous work done under the guidance of Dr. S K Sahay and using Naïve Bayes classifier to classify malwares. The previous approach aims to divide the malwares and benign into groups based on their sizes and then calculate the mean, variance for each group to prepare training model for the classifier.

The trained model is then used to classify the known and unknown malwares and benign based on their opcode frequencies. Their work provides an accuracy of 97.18%. The dataset used for experimental analysis consist of 11088 malware samples from malacia-project and 4006 benign programs from different systems.

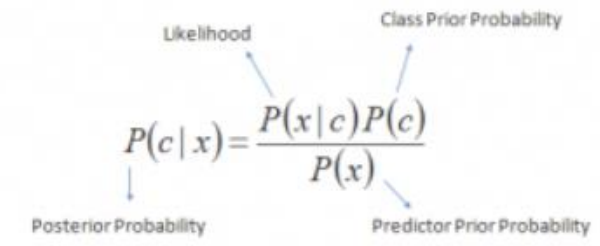
## CHAPTER 5

# NAÏVE BAYES

### 5.1 Introduction

It is a supervised classification technique based on Bayes Theorem with an assumption of independence among predictors. According to it all properties under consideration independently contribute to the probability for the decision making for classification. This model is easy to implement and useful for very large dataset. It is known to outperform even highly sophisticated classification methods.

Byes theorem helps in calculating posterior probability  $P(c|x)$  from  $P(c)$ ,  $P(x)$  and  $P(x|c)$



The diagram shows the Bayes' Theorem formula  $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$  with arrows pointing to its components: 'Likelihood' points to  $P(x|c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c|x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

$P(c|x)$ : - Posterior probability of target class  $c$ , given predictor attribute  $x$

$P(c)$ : - Prior probability of class  $c$

$P(c|x)$ : - Likelihood, which is the probability of predictor given class.

$P(x)$ : - Prior probability of predictor

## **5.2 Feature Selection**

Classification techniques needs some criteria for dividing or separating input into different classes. The process of identifying certain properties based on which that criteria is based on is called feature selection. Feature selection helps us in decreasing training time, simplify model construction and interpretation and removal of irrelevant and redundant features.

## **5.3 Grouping**

In previous work the grouping of Malware files in groups of 5KB file size has given good classification efficiency for Naïve Bayes classifier. The size of 5KB is considered because the Malwares samples generated using various toolkits has a size difference of less than 5KB. It has been shown in their work that there was an improvement of 8% in classification due to this method so as an extension of the previous work we also stick to that criteria.



## CHAPTER 6

### CURRENT WORK

This work is an extension of the previous work so the above mentioned dataset was used for this work also

#### 6.1 Implementation

##### Step1:

Understanding the CUDA architecture and CUDA programming model.

##### Step2:

Understanding the previous work and analyzing the implementation details.

##### Step3:

Bug fixing in the existing implementation and recompiling the whole work for better understanding

##### Step4:

Separating the existing parallel and serial structures from the existing implementation

##### Step5:

Rewriting the existing code flow and providing it a modular architecture in sequential form. This step also includes dumping various intermediate logs and model that are used in testing phase. This step was necessary to make the

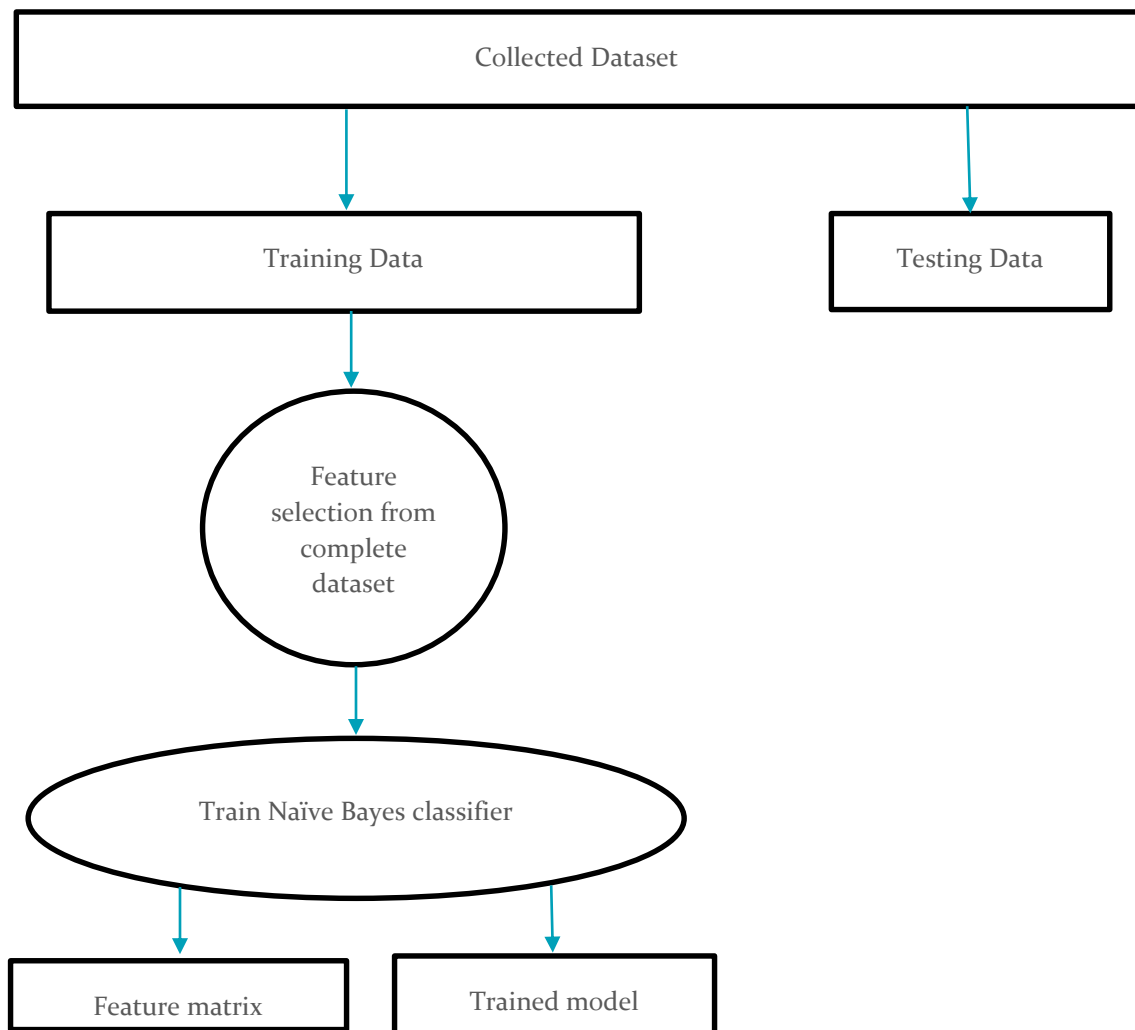
testing part independent of training part and saves the time of retraining at every testing request thus optimizing the overall detection time.

Along with the above mentioned modification now this implementation can take variable number test files as input.

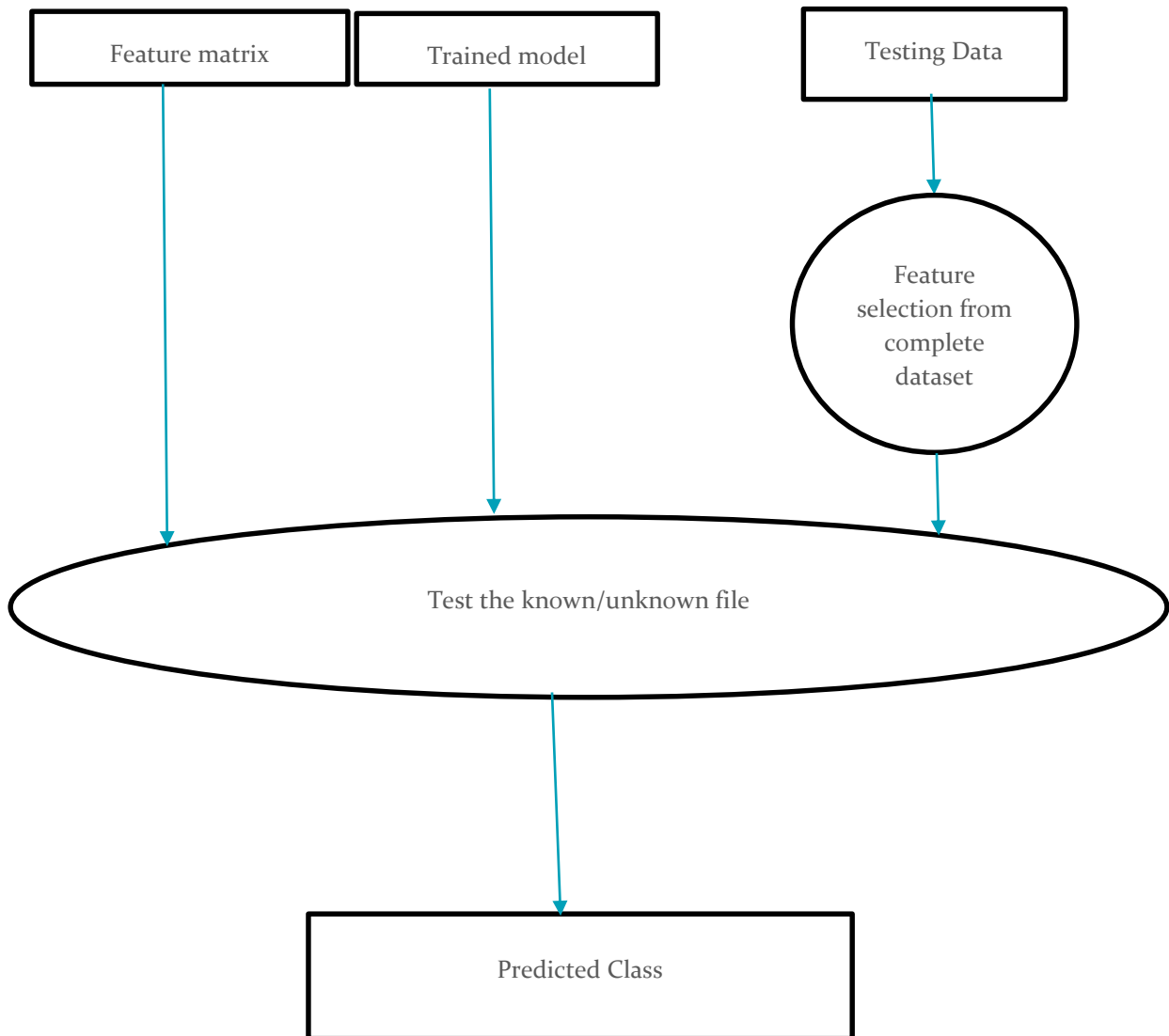
#### Step6:

Implementing the testing module in parallel fashion over GPU for increased timing efficiency.

### 5.3 Training module



## 5.4 Testing module



The pseudo code can be found in Appendix section.

## 5.4 Current status

The present work is able to achieve the successful parallel implementation of Naïve Bayes Classifier for malware detection.

The current work under progress is to understand the anomaly in the execution timing's due to memory access patterns. We are using some profiling tools and programming methods to check the current behavior and further improve the speedup of the process. The current implementation is able to maintain the same accuracy of previous work and able to achieve a maximum speedup of 52%. The verification work is still under progress and we hope to achieve a better performance then it.

## **CHAPTER 6**

### **CONCLUSION AND RESULTS**

#### **6.1 RESULTS AND FUTURE WORK**

The serial implementation of work takes 374 milliseconds for the execution of 13568 files whereas the GPU counterpart is able to execute in 9.01 milliseconds. At first the CPU implementation gives good results due to faster clock speed but with increase of samples the GPU starts dominating. Another finding was that the GPU execution time increases marginally after all the cores are being fully utilized. We are able to achieve an average speed up of 45 times than CPU using GPU while the maximum speedup obtained was 53 times of CPU. In addition to the speedup we were able to maintain the previous accuracy.

This work can be extended to full featured framework for malware detection equipped with a very good accuracy and execution time. This model can be further improved with integrating other learning and classification algorithms.

#### **6.2 CONCLUSION**

The use of GPGPU for malware detection is a new concept and the massive computation power of GPU can be exploited to improve the execution time of malware detection. This has been proved I our results in which we obtained a maximum speedup of 52%.

The involvement of machine learning techniques can come handy and tackle both known known and unknown malware threats if deployed with proper knowledge. The Naive Byes is the simplest and easy algorithm to start with and provides good results in detecting the malwares.

## **BIBLIOGRAPHY**

- [1] NVIDIA. CUDA C best practice guide. (March), 2011
- [2] Ashu Sharma, Sanjay K Sahay, and Abhishek Kumar. Improving the detection accuracy of unknown malwares by portioning the executables in groups.
- [3] NVIDIA, CUDA by example, (July), 2010
- [4] Nwokedi Idika, Aditya P.Mathur, A survey of malware detection techniques.

# APPENDIX

## Training Module

Input: malwares\_csv, benign\_csv, opcode\_list

Output: trained\_matrix\_csv, feature\_matrix\_csv

start

filelist[] = null

grouplist[] = null

**for** each **file** in input

    add to filelist

    group\_no = filesize/5

    add to grouplist

        if malware add to grouplist.maleare

        else add to malware grouplist.benign

**for** each file in input // normalization

    nopcode\_file[i] = (float)(opcode[i].freq - min)/(float)(max - min);

**end for**

// train naïve Bayes

**for** each file in training set

    class = file.class

    nfcc = # of files in current group

**for** each opcode in file.opc

        opcodeindex = opcode.index

        mean = file.opc[opcodeindex].norm\_value/nfcc

        variance = (mean-file.opc[opcodeindex].norm\_value)^2/nfcc

    trained\_matrix[class][mean][opcodeindex] += mean

    trained\_matrix[class][variance][opcodeindex] += variance

**end for**



end for

end

## Testing module

Input: test\_file.csv, trained\_matrix.csv, feature\_matrix.csv

Output: class

start

malware\_prob = 0

benign\_prob = 0

for each opcode opc

    if(test\_file\_vect[opc] > 0)

        value = test\_file\_vect[opc]

        malware\_prob += \

            getProb(mean\_var\_data[malware][opc].mean, mean\_var\_data[malware][opc].var, value)

        benign\_prob += \

            getProb(mean\_var\_data[benign][opc].mean, mean\_var\_data[benign][opc].var, value)

    end for

if (malware\_prob > benign\_prob)

    assign malware

else

    assign benign

end