DETECTING METAMORPHIC VIRUSES USING PROFILE HIDDEN MARKOV MODELS

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Computer Science

By

Srilatha Attaluri

December 2007

Approved by: Department of Computer Science
College of Science
San José State University
San José, CA

Dr. Mark Stamp

Dr. Chris Pollett

Dr. Agustin Araya

ACKNOWLEGEMENTS

I would like to thank Dr. Mark Stamp, for his guidance, encouragement and patience through out the project. My gratitude to Dr. Chris Pollett and Dr. Agustin Araya, for their valuable suggestions and feedback. My special thanks to Dr. Sami Khuri for introducing me to the amazing field of Bioinformatics and helping me understand Hidden Markov Models.

This project would not have been possible without the support of my family especially my loving husband, Satyadeva Prasad.

ABSTRACT

Detecting Metamorphic Viruses using Profile Hidden Markov Models

By Srilatha Attaluri

Metamorphic computer viruses “mutate” by changing their structure every time they propagate. Unlike other viruses, they use code obfuscation techniques on the body of the virus and do not exhibit a common signature. With the advent of construction kits, it is easy to generate various metamorphic strains of a virus.

Profile Hidden Markov Models (PHMM) are used in Bioinformatics for finding family-related DNA sequences. In this project we analyze and determine whether PHMM can be used to detect metamorphic virus family variants generated from three construction kits.

Each construction kit has a diverse behavior and hence different PHMM models must be generated by grouping a few strains of each construction kit. Models thus created hold opcodes probabilities calculated depending upon their occurrence in the virus variants. We then proceed to classify virus and non-virus files by scoring them against these models using Forward algorithm.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

The evolution of computer viruses shows that they are getting wittier everyday. Today's viruses target Internet websites to spread faster and further across the world. In earlier days, generating viruses required assembly language programming skills, but lately due to the arrival of various virus construction kits and mutation engines, any user with minimal or no knowledge of viruses can create lethal new strains of known viruses.

The most popular virus detection technique used today is signature detection, which looks for unique strings pertaining to known viruses. Once detected, a virus is no longer a threat if the signatures on the system are kept up to date. To bypass detection, virus writers started changing old viruses instead of creating new ones. This evolved into encrypted viruses that use a different key each time they propagate, but these often have a signature in their decryptors. Polymorphic viruses, on the other hand, started out using random encryption schemes and developed into decryptors' morphing. Although virus writers change the virus code significantly, most of these viruses can still be detected using signature detection when they are decrypted.

Metamorphic viruses alter the virus' entire code without changing its impact. Code obfuscation techniques like garbage code insertion, code reordering and sub-routine permutations are used to generate various variants that belong to a virus family. It is now easier to generate new metamorphic virus variants using construction kits, but detecting them is a challenge. Signature detection is not effective as each variant has a different scan string. Other anti-virus techniques like code emulation and heuristics can be used to detect them but are not time-efficient.

Hidden Markov Models are well-known for their use in speech recognition [4]. other applications include modeling protein sequences for protein families and patterns in RNA splice junctions [3]. Using Hidden Markov Models for detecting metamorphic viruses produced impressive results [9]. In this project we determine whether a special case of Hidden Markov Models, called Profile Hidden Markov Models (PHMM), can be used in detecting metamorphic strains of a virus.

Profile Hidden Markov Models are used in Bioinformatics for finding distantly-related sequences of a protein sequence family [1]. We focus on using PHMM to model a metamorphic virus family and score virus and non-virus files using the model. A PHMM model contains a group of probabilities and is created using an opcodes alignment of various virus family variants. We then proceed to differentiate virus and non-virus files depending upon their relativity to the model that is measured using Forward algorithm.

The report is organized as follows:

- Section 2 contains information about the evolution of metamorphic viruses and virus construction kits.
- Section 3 details a few code obfuscation techniques that are used for generating metamorphic variants.
- Section 4 describes the algorithms and theory of Profile Hidden Markov Models.
- Section 5 discusses various anti-virus technologies currently used.
- Section 6 provides a detailed discussion of test data generation, implementation details of training a PHMM model and scoring virus/non-virus files against the model.
- Section 7 provides results including detection, false positive and false negative rates.
- Section 8 draws conclusions based upon these findings.
- Section 9 discusses additional future enhancements.

2. METAMORPHIC VIRUSES

2.1 Origin of Viruses

Viruses started out as self-replicating programs at universities to spite other students, but these were mostly harmless. Although viruses were known to exist in the early 1980’s, during the time when personal computers arrived, they became popular for their malicious activities in 1988 with the advent of the Morris worm. Worms propagate by themselves, but viruses need help to spread. Robert T. Morris, jr., the author of the Morris Worm, used the Internet to spread and infect as many systems as possible. It brought the whole Internet to a halt with a denial of service attack that created widespread panic and awareness of viruses. Other viruses that were around at this time like Leigh, Brain and Jerusalem, targeted files, boot sectors or applications. Some of the viruses that emerged in the late 1980’s and early 1990’s had a payload associated with them. The destructive behavior of the virus is triggered when the payload conditions are satisfied.

One of the main objectives of a virus, apart from causing damage, is to remain undetected from anti-virus programs. Signature detection is a popular anti-virus technique that is used in detecting these viruses (more about it is discussed in Section 5.1). Writing new viruses from scratch is difficult and time consuming, hence most of the virus writers try to enhance existing viruses by fixing their bugs and making them more evasive. This may not change the signature of the parent virus, thus making them still detectable.

To bypass the detection, virus writers started hiding and changing the virus code. Encrypting the viruses changed them, but they had a signature in their decryption block. But signatures taken from decryptors can lead to flagging non-viruses that contain similar decryption blocks, increasing the false positives. Other complex cases include non-linear decryption and exclusion of decryption code from the virus. Oligomorphic viruses go a step further by dividing their decryptors into multiple parts or by instruction reordering. The changes in oligomorphic virus copies are subtle but still contain a constant string to search for.

So how to make the decryptors look very different from one another? The answer lies in polymorphism. Polymorphic viruses mutate their decryptors using code obfuscation techniques like garbage code insertion and equivalent code submission (code obfuscation techniques are discussed in detail in Section 3). Obfuscation and multiplayer encryption can generate millions of copies and hence each new generation creates a new polymorphic virus strain. In 1990 Mark Washburn wrote the first known polymorphic virus, “1260,” which uses garbage code insertion to vary its decryptor’s length [11]. Polymorphic viruses seem to interest the virus writers, as there are more of them than any other viruses today.

The main disadvantage of polymorphic viruses is that the body of the virus is not changed, so irrespective of their complexities, they can be detected by decrypting them using an emulator. Although emulating and decrypting them may be tedious, it is not impossible. Some of the viruses developed today employ anti-emulating techniques like unnecessary calculations, but an experienced debugger could overcome this. Can we mutate the virus itself instead of mutating its decryptors? This is exactly what a "metamorphic" virus does. A metamorphic virus obfuscates the entire virus body, thus forming millions of variations of the same virus.

2.2 Metamorphic Viruses

Metamorphic viruses usually use multiple obfuscation methods, giving them more variations. The degree of the mutation depends upon the section of the code that deals with morphing, called the metamorphic engine. A good metamorphic engine uses at least two of the code-obfuscation methods. Obfuscation methods range from simple register renaming to advanced code-substitution methods. More about obfuscation is discussed in section 3. Some of the methods, apart from obfuscation, also use encryption to generate completely different strands of viruses. Metamorphic engines are hard to write. One of the virus writers, "Benny," agrees to its complexity, and makes an incomplete metamorphic engine free to download.

32-bit metamorphic viruses infected systems that use window's 32-bit platforms and caused more damage than their earlier DOS-based siblings like TMC. Regswap in 1998 swapped registers in its variants but the actual source code was not changed, rendering it not very metamorphic. Win32.Apparition is known to be the first 32-bit metamorphic virus that appeared in early 2000. It uses garbage code insertion to generate variants. An affected system automatically emails the passwords to its creator, and infected files are corrupted when an attempt is made to remove the virus. It is still marked as critical even though it was launched seven years ago [20].

W32.Evol emerged in the middle of 2000, with a metamorphic engine that could generate a fixed number of variants combining the concepts of garbage and equivalent code substitutions. Unlike most of the viruses that infect all exe files, Evol targets only application exe's that are large enough to accommodate its code and do not use exports [21]. A signature is perceived on the execution stack but not in the code, which makes it hard to detect through heuristics and string scanning. Obfuscation rules are efficacious and are selected at random while generating new strains of Evol viruses.

Other advanced metamorphic viruses like Zmist and Win32.Metaphor have randomly selected many methods including on-the-fly encryption and attacks depending upon the structure of the infected file. Vecna, a member of 29A virus writing group, started creating viruses in the early 90's and came up with "Lexotan32" in 2002. Lexotan32 overcomes the problem of creating new variants by maintaining a table that helps in de-permuting the code and regenerating the new obfuscated code combining many techniques known in metamorphism [22].

Metamorphism is different from permutation, permutation deals with reordering the code but metamorphism substitutes Permutation viruses like Zperm and Bistro scramble their instructions to change their memory stamps. Permutation may not hide the signatures, but when coupled with code morphing it produces unrelated variants. Consider a program with two subroutines (X_0 and Y_0) and two variants per subroutine (X_1, X2, Y_1 & Y_2). Assuming that a signature exists at a point where the subroutines merge (so the order in which they appear is important), there would be 17 variations that would miss a signature based on one variant. Fortunately virus writers cannot predict the signature and need to use complex methods for a true metamorphic copy.

Mutation engines, on the other hand, help to change the virus structure instead of creating destructive code themselves. There are a wide variety of these engines for jobs like decryptor permutation, code compression, anti-heuristics, code permutation and metamorphism. Mutation engines work as black boxes, taking an existing virus as input and outputting a totally new variant. Most of them work on expanding, shifting and shrinking the existing code and are very effective in cheating signature detection. Zombie’s Code Mutation Engine (ZCME) is an example of a metamorphic engine that uses its own disassembler to get the source code and then changes the original code by randomly shuffling the code like changing the jump instructions and adding “nop” instructions. Other metamorphic engines, like Simile and MSIL metamorphic engines, as discussed in [11] by Peter Szor, emphasize the capability of mutation engines.

The most recent metamorphic viruses were seen back in 2002, indicating that virus writers seem to be concentrating more on spreading them rather than developing new ones.

2.3 Construction Kits

Web sites like VXHeaven give the source code for viruses and obfuscation engines, enabling novice writers to develop advanced viruses. But interested users need a minimum of assembly language programming skills to combine them into a metamorphic virus. Construction kits combine features like encryption and anti-debugging with metamorphic/polymorphic engines, allowing even a normal computer user to generate deadly viruses. Some of the kits are capable of generating thousands of new variants.

Construction kits are available for viruses, trojans, logical bombs and even worms. Since they create several variants with ease, it poses a considerable challenge to the anti-virus vendors. We have used a few construction kits like virus-creation library, phalcon-skism and next generation virus creation kit for our project. As different programmers developed these kits, it gives us a chance to see the performance of Profile Hidden Markov Models in detecting them.

Following is a brief description of each of the virus construction kits used in the project:

- Virus Creation Lab (VCL32) creates win32 virus variants depending upon user preferences. The first version of VCL, as created by a group of virus writers called NUKE, came around 1992, and a newer version developed by another group, “29A,” surfaced in 2004. Unlike other construction kits that use the command prompt for generating variants, it provides a GUI to choose from various preferences. Preferences that can be changed include which section of the host to infect, network or current directory infection, message box data, etc. VCL can also be set to use either a polymorphic engine or the KME-32 mutation engine that mutates decryptors.

  Once the options are chosen, VCL generates assembly language code files of the virus strains. These files can later be compiled and linked to get the exe files. It has been reported that the code generated by the earlier version had bugs and could not be compiled, but the newer version seems to have overcome those problems. We have used Borland Turbo Assembler and Tools (TASM) version 5.0 to compile and link. Many virus creators recommend TASM over Microsoft Assembler (MASM) to compile their assembly sources.

- Phalcon-SKISM group, a competitor to VCL’s NUKE GROUP, created Phalcon/Skism Mass-Produced Code Generator (PS-MPC). Phalcon and SKISM merged to form Phalcon-Skism group [19]. Unlike the first version of VCL, PS-MPC performed well in creating serviceable viruses. A configuration file is used to change the settings with around 25 alternatives that include optional parameters like payload. A kit user has a choice between infecting COM and exe files, memory resident and null encryption. Payload depends upon the month, day and time specified in the virus, as well as minimum or maximum file sizes to infect. PS-MPC also implements obfuscation of the decrypting section, but it does not implement other virus techniques like anti-debugging and anti-emulation techniques.

- Next Generation Virus Creation Kit (NGVCK), created by SnakeByte, surfaced in 2001 and, as far as we know, is by far the most advanced virus constructor. Unlike VCL and PS-MPC there is no need to set configuration settings as it automatically generates a new variant every time it is used. This construction kit implements code obfuscations like junk code insertion, subroutine reordering, random register swapping and code-equivalent substitutions. NGVCK is developed as a non-virus program with multiple revisions and beta versions. We have used version 30 as it is said to be stable and more advanced than its siblings. The NGVCK kit is programmed to satisfy the needs of both novices and advanced programmers. Advanced programmers can select the kind of encryption, anti-tricks and directory traversal.

Following is a small example given in the introduction document distributed along with the kit, explaining the kind of obfuscations it implements:

Basic Version	Morphed Version 1	Morphed Version 2
call Delta Delta: **pop** ebp **sub** ebp, offset Delta	**call** Delta Delta: **sub** dword ptr[esp], offset Delta **pop** eax **mov** ebp, eax	**add** ecx,0031751B ; junk **call** Delta Delta: **sub** dword ptr[esp], offset Delta **sub** ebx,00000909 ; junk **mov** edx,[esp] **xchg** ecx,eax ; junk **add** esp,00000004 **and** ecx,00005E44 ; junk **xchg** edx,ebp
Hex equivalent: E8000000005D81ED05104000	Hex equivalent: E800000000812C2405104000588BE8	Hex equivalent: *812C240B104000*8B1424*83C404*87EA

Table 1: Code Obfuscation Example for NGVCK

In Table 1, morphed versions show the obfuscated code of the basic version. Morphed version 1 uses obfuscations like code reordering and equivalent code substitution, whereas version 2 also uses junk code insertion. The hexadecimal equivalents shown are very different and signature scanning is clearly not a solution.

Apart from code obfuscation it also implements anti-debugging and anti-emulation techniques to hide from the anti-virus researchers. Unlike metamorphic engines that create variants from a given source code, NGVCK morphs the source code itself to create variants. The programmer has tried to create a 100% variability between different strains; the later versions were targeted to add more layers of encryption and morph the decryptors.

Construction kits and mutation engines are here to stay for their ease of use and personalization of new viruses, but are extremely deadly as they can resurrect different strains of age-old viruses. Such morphing of old viruses would reopen the same problems anti-virus once had, so it is very important to use machine-learning techniques and some kind of automation to detect them.

3. CODE OBFUSCATION TECHNIQUES

Code obfuscation is transforming the code and making it obscure or difficult to understand [6]. Software programmers use these techniques to make their product resistant against reverse engineering. Metamorphic virus writers use one or more of these techniques to create a unique copy of existing virus, which makes them indistinguishable to virus scanners.

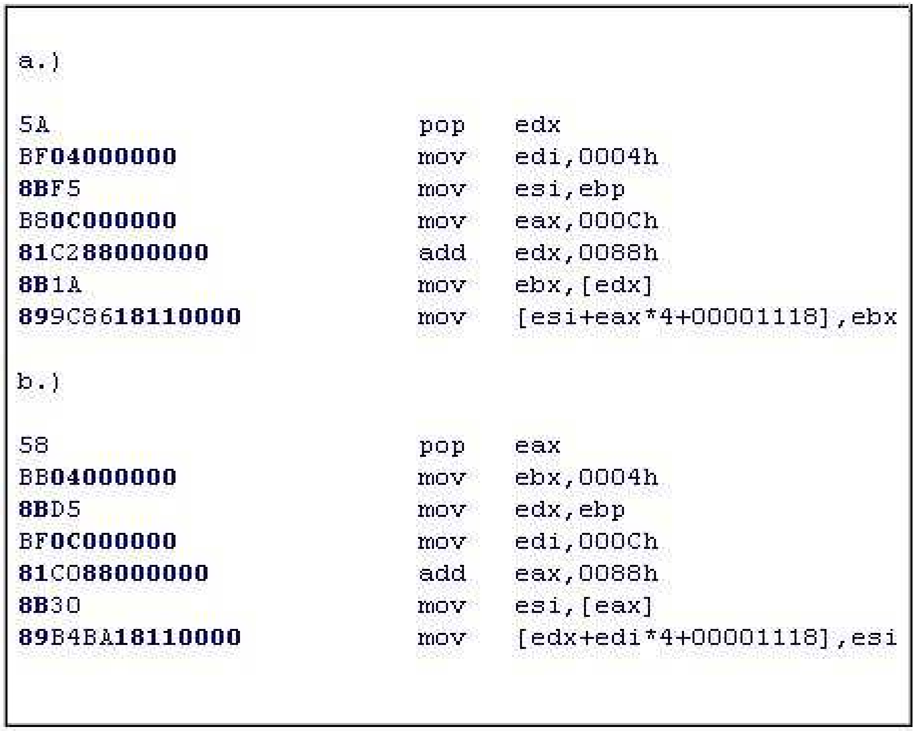
3.1 Garbage Code Insertion

Garbage or do-nothing codes are programming instructions that are a part of the program physically but not logically. They are not related to the program's outcome. Do-nothing instructions such as register exchanging (XCHG) slow down code emulation. Other instructions such as “NOP”,”MOV ax, ax”, ”SUB ax, 0”, etc make the virus look different and thus possibly escape heuristic analysis. Garbage instructions may also be branches of code that are never executed or which have some calculations done on the variables declared in other garbage blocks. The main idea of this code obfuscation technique is to confuse and exhaust the virtual machine or person traversing the virus code.

However, the virus scanners these days are powerful enough to get past these do-nothing instructions. When there are too many of such instructions perceived in a file it may be flagged as a virus because it is highly unlikely there would be such instructions in non-virus programs.

3.2 Register Renaming

‘Register renaming’ is modifying the names of variables or registers used in a virus. When registers are changed they result in different opcodes that trick the signature search. Regswap is a metamorphic virus that swaps the registers for each variant.



```
a.)

5A                        pop    edx
BF04000000                mov    edi,0004h
8BF5                      mov    esi,ebp
B80C000000                mov    eax,000Ch
81C288000000              add    edx,0088h
8B1A                      mov    ebx,[edx]
899C8618110000            mov    [esi+eax*4+00001118],ebx

b.)

58                        pop    eax
BB04000000                mov    ebx,0004h
8BD5                      mov    edx,ebp
BF0C000000                mov    edi,000Ch
81C088000000              add    eax,0088h
8B30                      mov    esi,[eax]
89B4BA18110000            mov    [edx+edi*4+00001118],esi
```

Figure 1: Regswap Variants [11]

Two variants of regswap shown in Figure1 have the same set of instructions but use different registers. If these instructions form the signature, the virus succeeds in bypassing detection. For detecting such viruses a signature should not be over fitting and be like a regular expression that can overcome register changes with wild characters [11].

Memory traces are the key in analysis of unknown viruses. Among the other code obfuscation techniques, register renaming benefits the creator by having different memory traces for each of its variants.

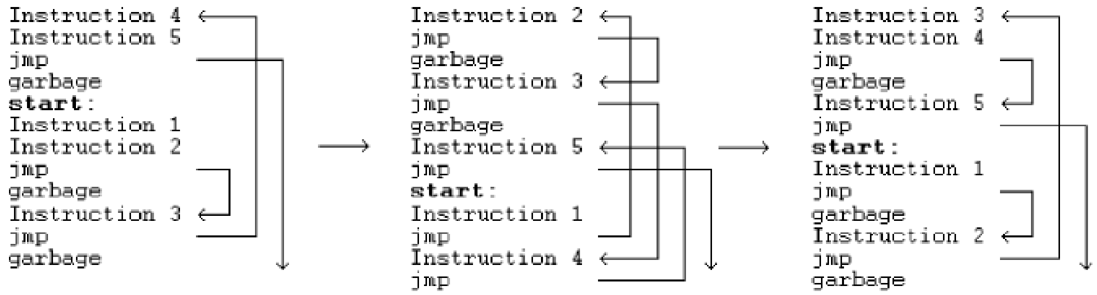
3.3 Subroutine Permutation

Subroutine permutation is a simple obfuscation method where the subroutines are reordered. It will not affect the impact of the virus, as the order in which subroutines appear in the code is insignificant to a program's execution. Thus a virus containing 'n' subroutines can have 'n!' permutations. Compared to the other obfuscation methods, subroutine permutation can be easily detected by signature detection, as the signature still exists in clear view. Metamorphic viruses like Win95.Ghost and Win95.Smash are examples of this behavior [20].

But rearranging subroutines poses considerable challenges to some of the analysis methods. This project models a given virus family from a multiple sequence alignment, which is obtained by arranging multiple sequences depending upon a matched region of opcodes. If a program is permutated, most of the regions do not match, giving a weak alignment and hence a weaker model. A solution to this obfuscation is to de-permute each sequence before aligning them.

3.4 Code Reordering through Jumps

Code reordering alters the order of the instructions but maintains the original instruction's logical flow using jumps. Reordering the code creates control flow obfuscation as the control changes depending upon unconditional jumps. These unconditional jumps are inserted randomly, challenging its detection by memory mapping.



```
Instruction 4
Instruction 5
jmp
garbage
start:
Instruction 1
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
```

```
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
Instruction 4
jmp
```

```
Instruction 3
Instruction 4
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
garbage
Instruction 2
jmp
garbage
```

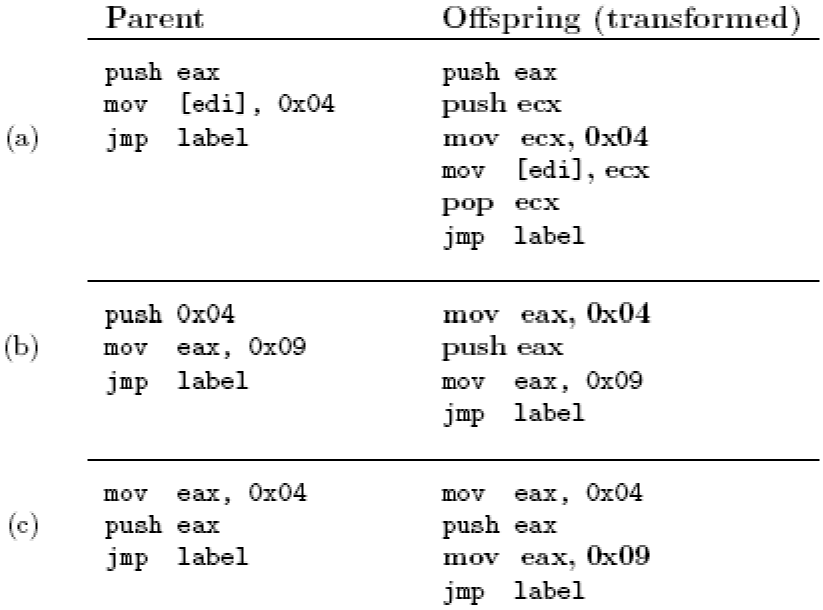
Figure 2: Code Reordering [7]

Figure 2 shows an example of code reordering. This fairly simple method overcomes signature detection by altering the signature-bearing opcodes sequence.

3.5 Equivalent Code Substitution

Each task can be done in different ways. Similarly, virus codes, although looking different, can accomplish the same task. Substitution of equivalent codes for virus codes escapes few detection techniques. It can be caught through behavior checking since the execution does not change in many cases.

This type of obfuscation can also be used to shrink or expand the original code by substituting the code with smaller or larger equivalent codes. As a simple example "**ADD ax, 3"** can be transformed to "**SUB ax, -3"**, as both the instructions add a 3 to the contents of ax register. It can also be accomplished with a two-step process like **"MOV bx, -3"** and **"SUB ax, bx"**. W32.Evol is a metamorphic virus that randomly substitutes equivalent code, generating different strains in each generation, Figure 3 shows a few substitutions perceived in this virus [18].



	Parent	Offspring (transformed)
(a)	push eax mov [edi], 0x04 jmp label	push eax push ecx mov ecx, 0x04 mov [edi], ecx pop ecx jmp label
(b)	push 0x04 mov eax, 0x09 jmp label	mov eax, 0x04 push eax mov eax, 0x09 jmp label
(c)	mov eax, 0x04 push eax jmp label	mov eax, 0x04 push eax mov eax, 0x09 jmp label

Figure 3: Code Substitutions in W32.Evol Metamorphic Virus [18]

Each code segment in the offspring works exactly as its parent with little tweaks in the parent code. Often, mutated code is not simple enough to be detected by string search. However, variants shown in the above example can be detected using a wild string in the signature. One of the detection techniques used to tackle such advanced obfuscation is to transform the code into a simple code [12].

4. THEORY OF HIDDEN MARKOV MODELS

4.1 Markov Chains

Markov chains are a series of states with probabilities associated with each transition between states. Transition probabilities calculated from the current state are independent of its previous states [3].

A Markov chain for a DNA sequence is shown in Figure 4 [1]. DNA’s chemical code is an alphabet of four symbols called bases denoted by A (adenosine), C (cytosine), G (guanine) and T (thymine).

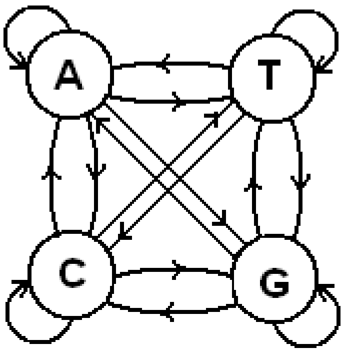


Figure 4: Markov Chain for DNA [1]

Each arrow in Figure 4 represents the transition probability of a base followed by another base. Transition probabilities are calculated after observing several DNA sequences. A transition probability matrix can represent these transition probabilities. The DNA Markov model is a first order Markov model since each event depends on its previous event.

The transition probability a_{st} (Transition Probability from a previous state with symbol s to current state with symbol t) is calculated as [1]:

$a_{st} = P(x_i = t |\ x_{i-1} = s) \quad 1 \leq s, t \leq N$ (N is the number of states)

The sum of the transition probabilities from each state is equal to 1. Since there is a probability associated with each step, this model is called as a Probabilistic Markov Model [10].

The Probability of a given sequence against a model is calculated as [1]:

$$\begin{aligned} P(x) &= P(x_L, x_{L-1}, \ldots. x_1) \\ &= P(x_L|x_{L-1}, \ldots x_1)\ P(x_{L-1}|x_{L-2}, \ldots.. x_1) \ldots. P(x_1) \\ &= P(x_L|\ x_{L-1})\ P(x_{L-1}|\ x_{L-2}) \ldots. P(x_2|\ x_1) P(x_1) \qquad \text{(using Baye's Theorem)} \\ &= P(x_1) \prod_{i=2}^{L} (a_{x_{i-1} - x_i}) \end{aligned}$$

$P(x_1)$ is the probability of starting at a state with symbol x_1. This can be calculated by adding a begin state, and an end state to accommodate first and last symbols of the sequence.

4.1.1 High Order Markov Chains

High order Markov chains are those in which the current event depends on more than one previous event. As defined in [1] “an nth order Markov process is a stochastic process where each event depends on previous n events". An nth order Markov process with an alphabet of m symbols can be represented as a first order markov chain with an alphabet of m^n symbols. Consider a two-symbol alphabet {A,B}. This is similar to the binary code, a sequence like ABAAB will be paired as AB-BA-AA-AB and can be represented by a four-state first order Markov model with states AB, BB, BA and AA.

4.2 Hidden Markov Models

Given a sequence and a markov chain, one could determine which state generated each symbol from the sequence, but in many cases this may not be apparent. Consider the urn and ball model stated in [4] by Rabiner in 1989. Assume that there are N glass urns with different colored balls in them as shown in Figure 2 (i.e. we know the probability of each ball in each urn), depending upon a process (that takes into consideration a previously-selected urn for selecting a current urn) some balls are picked. Now, given a sequence of balls picked, like {Red, Blue, Orange, Red…}, we do not know which urn was used to pick a particular ball in the sequence.

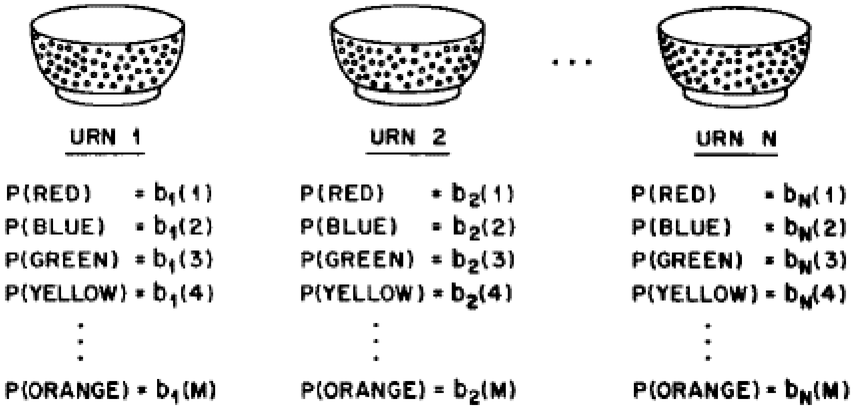


Figure 5: Urns and Ball Model [4]

So the unobserved or "hidden" process of urn selection is observed through the sequence of balls picked. Hidden Markov Models (HMM) are used for such problems. The main distinction between HMM and the Markov Chain is that in HMM given a sequence {x_1, x_2 …..x_i), it is not possible to tell which state generated a symbol x_i [1].

General notation used for HMM is [5]:
O - Observation sequence
T – Total number of symbols in the observation sequence
N - Total number of states

α - Alphabet for the model
M - Total number of symbols in the alphabet
π - Initial state distribution
A - State transition probability matrix
a_{ij} - Transition probability from state i to j
B - Symbol probability distribution matrix
$b_i(k)$- Probability distribution of k in state i
λ - HMM model
The HMM model is comprised of (A, B, π) along with N and M.

To help in understanding HMM better, consider an example where two coins--one biased, and one normal--are tossed T times to generate a sequence O by occasionally switching between the coins. The observed sequence is O = {HTHTHH} where H stands for heads and T for tails, giving the number of symbols in the alphabet {H,T} as 2 (M). The two states (N) in the model are Biased and Normal. Figure 6 depicts the model.

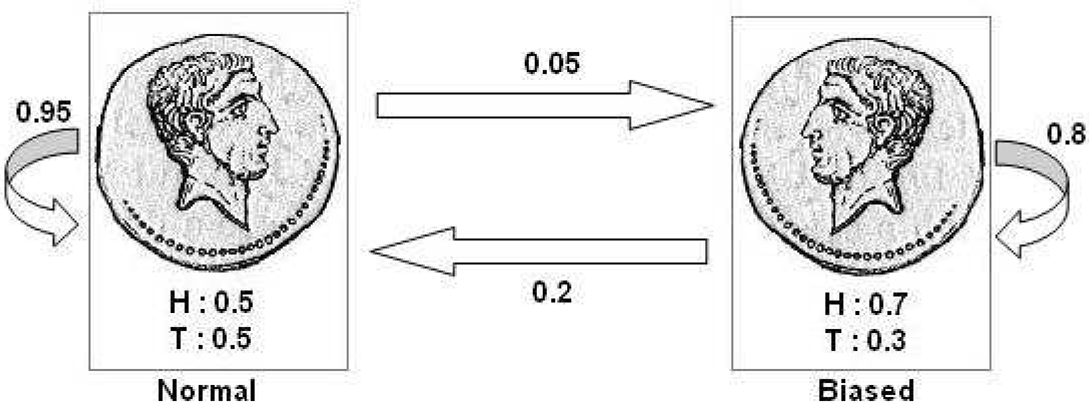


Figure 6 Example of HMM

The transition probability matrix taking Normal as 1 and Biased as 2, is as follows:

$$A = \begin{bmatrix} 0.95 & 0.05 \\ 0.2 & 0.8 \end{bmatrix}$$

i.e. a_{12} = 0.05 represents the transition probability to state 2 (Biased) from state 1 (Normal). The symbol distribution matrix (B) gives the probability distribution of H and T in both the states.

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix}$$

The first row gives the probability distribution of (H, T) in a Normal coin and second row is that of a biased coin. The representation $b_1(H)$ represents the probability distribution of

H in case of a Normal coin. The initial distribution determines which coin to start with; in this case it is taken at random.

$$\Pi = [0.5 \quad 0.5]$$

Hence the HMM model for the two-coin example is (A, B, π) with N, M also known. Notice that the sum of each row in the transition and symbol distribution matrices is 1.

The two-coin example is a fully connected HMM, also called as an ergodic model [4]. There are other types of HMMs, like left-right models with or without parallel paths. More detailed information on different types of HMM is given in [4].

4.2.1 Profile Hidden Markov Models

Multiple sequences of genes are combined to form an alignment that contains the hidden relation between them. A model created from the resultant multiple sequence alignment (MSA) is used to measure the relativity of an unknown sequence to a family. This idea is extended in our case where the sequences are opcodes of known metamorphic viruses.

These sequences can be represented by a large regular expression. However, such a model will be over-fitting and could miss other unknown mutations. Profile Hidden Markov Models (PHMM) are a type of HMM that profiles a given sequence alignment [3]. Unlike the HMMs seen so far, they allow null transitions, so that the model can also fit the divergent sequences. In the case of DNA, these divergences are caused during evolution [1]. Metamorphic viruses are, however, programmed to have these differences.

The basic advantage of profile HMM over HMM is that it is more useful in detecting distantly-related members of the family. The structure of a Profile HMM with the added null transitions and gaps in the sequence alignment looks like in Figure 7.

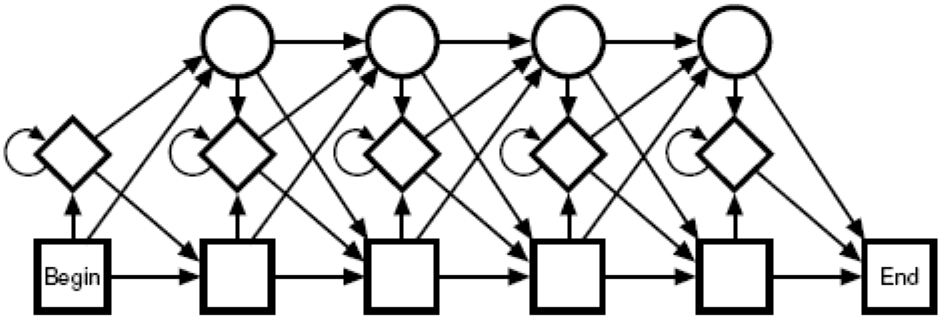


Figure 7 Structure of Profile HMM [2]

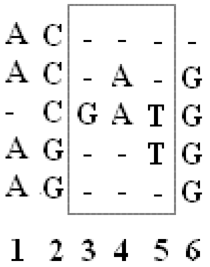
In Figure 7, circles that allow null transitions are called “Delete” states, diamonds that allow gaps in a sequence alignment are called the “Insert” states, and the rectangles are similar to the states in an HMM called “Match” states. Match and Insert states are the emission states of PHMM (i.e. whenever passed through these states, a symbol is emitted.) Emission probabilities are calculated depending upon frequency of symbols emitted. Delete states allow passing through the gaps found in MSA and reach other emission states.

The arrows in the figure represent the transitions possible from the current to the next state. Probabilities associated with them, called “Transition Probabilities,” determine the likelihood of the next state taken.

As in HMM, two states ‘begin’ and ‘end,’ are added to include the initial probability distribution for the first symbol and similarly to the last symbol of the sequence.

The general notation used in Profile HMM is similar to HMM:
X - Observation sequence
i – Total number of symbols in the Observation sequence $x_{1...i}$
N - Total number of states
α - Alphabet for the model
M – Match states $M_{1...N}$
I – Insert states $I_{0...N}$
D – Delete States $D_{1...N}$
π - Initial state distribution
A - State transition Probability Matrix
A_{kl} - transition frequencies from state k to l
$a_{M_1M_2}$ - Transition probability from state M_1 to M_2
E - Emission Probability Matrix for Match and Insert states
$E_m(k)$- Emission frequency of symbol k at state m
$e_{M_1}(k)$- Emission probability of symbol k at M_1
λ - HMM model

To understand profile HMM better, consider an example given the Multiple Sequence Alignment (MSA) obtained by sequences using the four bases of DNA as in Figure 4 (This sequence is merely an example and is not taken from any genuine biological sequences).



```
A C - - - -
A C - A - G
- C G A T G
A G - - T G
A G - - - G

1 2 3 4 5 6
```

Figure 8 Multiple Sequence Alignment Example

The first step in creating a Profile HMM model is to find which columns in the MSA form the match and insert states. One of the rules used as illustrated in [1] is to use the more conservative columns (i.e. at least more than half of the characters in the column are symbols) as the Match states and the others with more gap characters as Insert states. In the above MSA, the columns 1,2 and 6 become the Match states.

Next we start by calculating the emission probability for column 1, which results in:

$e_{M1}(A) = 4/4$ $e_{M1}(C) = 0/4$ $e_{M1}(G) = 0/4$ $e_{M1}(T) = 0/4$

It can be seen that most of these values are zero, but since the model is to be flexible we have to add small probabilities to other cases in order to incorporate all the cases that may arise. A simple rule to use is the “Add-one rule” [1] where we add 1 to the numerator and the total number of symbols in the alphabet to denominator e.g. $e_{M1}(A) = (4+1)/(4+4) = 5/8$.

This results in the following emission probabilities at Match states and Insert states:

$e_{M1}(A) = 5/8$ $e_{M1}(C) = 1/8$ $e_{M1}(G) = 1/8$ $e_{M1}(T) = 1/8$	$e_{I1}(A)$ = ¼ $e_{I1}(C) = 1/4$ $e_{I1}(G) = 1/4$ $e_{I1}(T)$ = ¼
$e_{M2}(A) = 1/9$ $e_{M2}(C) = 4/9$ $e_{M2}(G) = 3/9$ $e_{M2}(T) = 1/9$	$e_{I2}(A) = 3/9$ $e_{I2}(C) = 1/9$ $e_{I2}(G) = 2/9$ $e_{I2}(T) = 3/9$
$e_{M3}(A) = 1/8$ $e_{M3}(C) = 1/8$ $e_{M3}(G) = 5/8$ $e_{M3}(T) = 1/8$	$e_{I3}(A) = 1/4$ $e_{I3}(C) = 1/4$ $e_{I3}(G) = 1/4$ $e_{I3}(T)$ = ¼

Table 2: Profile HMM Emission Probabilities for the MSA in Figure 8

The general formula that can be used to calculate the emission probabilities is:

$e_n(k)$ = (Number of Occurrences of k in state n)/(Total number of symbols in state n)

The Emission Probabilities matrix (E) of PHMM is a little different from the symbol transition probability matrix (B) in HMM , since we have more than one way a symbol is emitted (match and insert).

Transition probabilities calculation is the next step in profile HMM modeling, and the general equation used in calculating it is [1]:

a_{mn} = (Number of transitions from m to n)/(Total number of transitions from m to any state)

$a_{BM_1} = a_{BM_1}/(a_{BM_1}+ a_{BI_0}+ a_{BD_1}) = 4/(4+0+1) = 4/5$

To avoid underflow while scoring a given sequence we use the add-one rule on transition probabilities e.g. $a_{BM_1} = (4+1)/(5+3) = 5/8$

$a_{BM_1} = 5/8$ $a_{BI_0} = 1/8$ $a_{BD_1} = 2/8$	$a_{I_0M_1} = 1/3$ $a_{I_0I_0} = 1/3$ $a_{I_0D_1} = 1/3$	
$a_{M_1M_2} = 5/7$ $a_{M_1I_1} = 1/7$ $a_{M_1D_2} = 1/7$	$a_{I_1M_2} = 1/3$ $a_{I_1I_1} = 1/3$ $a_{I_1D_2} = 1/3$	$a_{D_1M_2} = 2/4$ $a_{D_1I_1} = 1/4$ $a_{D_1D_2} = 1/4$
$a_{M_2M_3} = 2/8$ $a_{M_2I_2} = 4/8$ $a_{M_2D_3} = 2/8$	$a_{I_2M_3} = 4/8$ $a_{I_2I_2} = 3/8$ $a_{I_2D_3} = 1/8$	$a_{D_2M_3} = 1/3$ $a_{D_2I_2} = 1/3$ $a_{D_2D_3} = 1/3$
$a_{M_3E} = 5/6$ $a_{M_3I_3} = 1/6$	a_{I_3E} = ½ $a_{I_3I_3}$ = ½	$a_{D_3E} = 2/3$ $a_{D_3I_3} = 1/3$

Table 3: Profile HMM Transition Probabilities for the MSA in Figure 8

The final model for the MSA in Figure 8 with beginning and ending states added looks as shown in Figure 9.

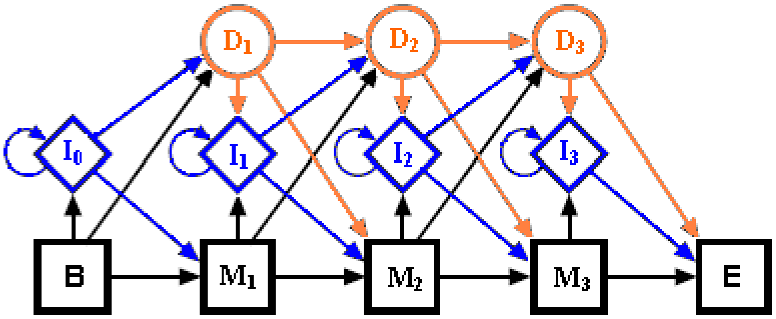


Figure 9: Profile HMM model

The final PHMM model for the MSA consists of E (emission probability matrix) with emission probabilities of Match and Insert states (Table 2) and A (Transition probability matrix) containing transitions from each Match, Insert and Delete states (Table 3) and the number of states including beginning and ending states (N) is 4.

4.3 Algorithms for Scoring Unknown Sequences against a Known Model

There are three basic problems in Hidden Markov Models as discussed in [4]:

Problem 1: Given a Model $\lambda = (A,B, \pi)$ and an observation sequence (X where $X = x_1 \ldots x_T$), how can we efficiently compute $P(X| \lambda)$ (i.e. the probability for the model to produce the observed sequence)?

Problem 2: Given a Model (A,B, π) and an observation sequence (X), how can we find the "correct" or optimal sequence of states which produce the given observed sequence?

Problem 3: How can the model (A,B, π) be changed to best fit the observed sequence?

4.3.1 Forward Algorithm

Forward Algorithm solves the first problem but before going there, let us see how $P(X| \lambda)$ can be calculated (i.e. the "inefficient" way). $P(X| \lambda)$ is interpreted as probability of the sequence X emitted by model λ.

The brute-force approach to calculate $P(X| \lambda)$ is taking the sum of probabilities of all possible paths to emit sequence X. For example, a sequence X = (A, B) emitted by a 4-state PHMM model takes 13 possible paths as shown in Table 4. A symbol is emitted each time they pass through an Insert or a Match state.

	I_0	I_1	I_2	M_1	M_2
1	A,B	-	-	-	-
2	A	B	-	-	-
3	A	-	B	-	-

4	A	-	-	B	-
5	A	-	-	-	B
6	-	A,B	-	-	-
7	-	A	B	-	-
8	-	A	-	-	B
9	-	-	A,B	-	-
10	-	B	-	A	-
11	-	-	B	A	-
12	-	-	-	A	B
13	-	-	B	-	A

Table 4: Possible Paths for a Sequence with 2 elements Emitted by a 4-state PHMM Model

Figure 10 shows the possible path traversals listed in Table 4.

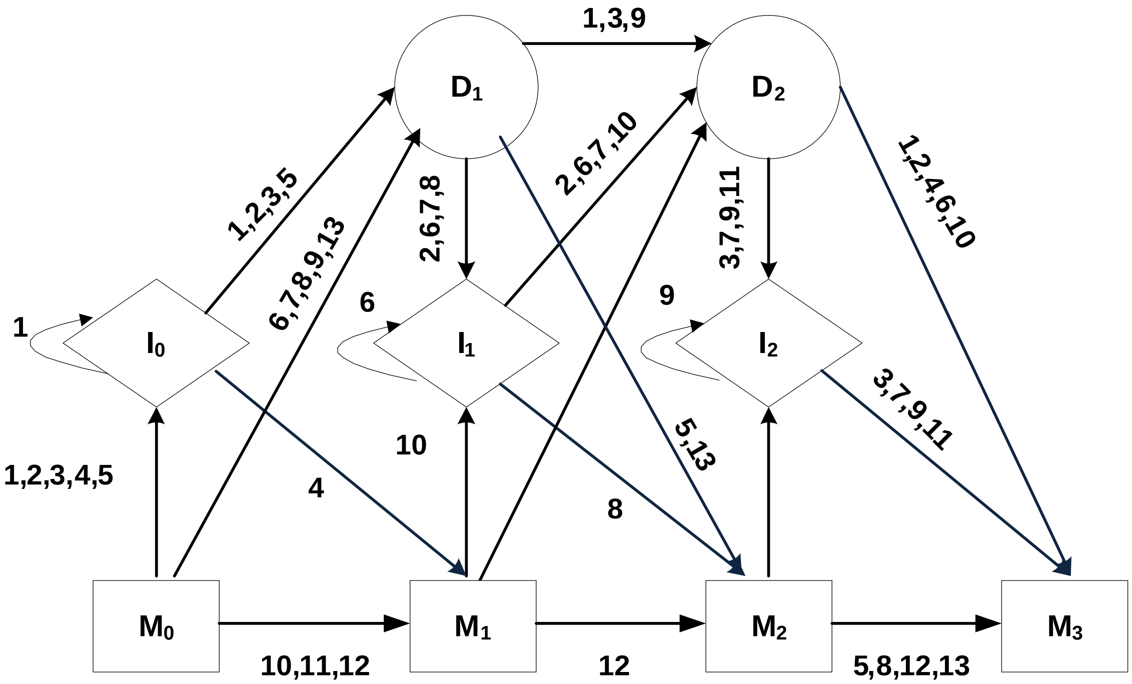


Figure 10: PHMM with 4 States Illustrating Emissions of a 2-element Sequence

Calculating probabilities for each of these cases is definitely not efficient. Forward algorithm computes the probability by reusing the already-calculated forward score of a partial sequence (i.e. at each level we consider the next states since we have the scores for the previous states already calculated). For a profile Hidden Markov Model the forward algorithm recursive relation is [1]:

$$F_j^M(i) = \log \frac{e_{M_j}(x_i)}{q_{x_i}} + \log \left[\begin{array}{c} a_{M_{j-1}Mj} \exp(F_{j-1}^M(i-1)) + a_{I_{j-1}Mj} \exp(F_{j-1}^I(i-1)) \\ + a_{D_{j-1}Mj} \exp(F_{j-1}^D(i-1)) \end{array} \right]$$

$$F_j^I(i) = \log\frac{e_{I_j}(x_i)}{q_{x_i}} + \log\left[\begin{array}{c} a_{M_jIj}\exp(F_j^M(i-1)) + a_{I_jIj}\exp(F_j^I(i-1)) \\ + a_{D_jIj}\exp(F_j^D(i-1)) \end{array}\right]$$

$$F_j^D(i) = \log\left[a_{M_{j-1}Dj}\exp(F_{j-1}^M(i)) + a_{I_{j-1}Dj}\exp(F_{j-1}^I(i)) + a_{D_{j-1}Dj}\exp(F_{j-1}^D(i))\right]$$

The base case for this recursion is $F^M_0(0) = 0$.

In the above equation, $F^M_j(i)$ represents the Forward score of subsequence $x_1 \ldots x_i$ up to state j. The background distribution is q_{xi} (distribution of symbol x_i in a random model).

During recursion, some insert and delete terms are not defined like $F^I_0(0)$, $F^D_0(0)$ … such items are to be ignored while calculating the scores. It can be seen that $F^M_j(i)$ is calculated as a function of $F^M_{j-1}(i-1)$, $F^I_{j-1}(i-1)$ and $F^D_{j-1}(i-1)$ and their respective transition probabilities to reach the match state from its previous state to emit the symbol x_i and includes the emission probability of x_i at M_j. Similarly, since insert and delete states do not emit the emission probability, the term is removed for calculating $F^D_j(i)$. States M_0 and M_{N+1} represent "begin" and "end" states respectively, and like delete states they also do not emit.

4.3.2 Viterbi Algorithm

The coin example from section 2.2 gives an observation sequence that looks like (H,T,H,T…) but we do not know if the first H in the sequence is generated by the biased or normal coin; this was the hidden part. In the second problem stated above, we need to find this hidden part. The Viterbi algorithm does exactly this. This problem is called the decoding problem in speech recognition. Viterbi based on dynamic programming techniques finds the sequence that maximizes the $P(X|\ \lambda)$. It does so by taking the sequence of states that generates the maximum probability at each level.

For a profile Hidden Markov Model the Viterbi algorithm recursive relation is [1]:

$$V_j^M(i) = \log \frac{e_{M_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_{j-1}^M(i-1) + \log(a_{M_{j-1}M_j}), \\ V_{j-1}^I(i-1) + \log(a_{I_{j-1}M_j}), \\ V_{j-1}^D(i-1) + \log(a_{D_{j-1}M_j}); \end{cases}$$

$$V_j^I(i) = \log \frac{e_{I_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_j^M(i-1) + \log(a_{M_jI_j}), \\ V_j^I(i-1) + \log(a_{I_jI_j}), \\ V_j^D(i-1) + \log(a_{D_jI_j}); \end{cases}$$

$$V_j^D(i) = \max \begin{cases} V_{j-1}^M(i) + \log(a_{M_{j-1}D_j}), \\ V_{j-1}^I(i) + \log(a_{I_{j-1}D_j}), \\ V_{j-1}^D(i) + \log(a_{D_{j-1}D_j}); \end{cases}$$

The base case is $V^M_0(0) = 0$.

The basic difference with the forward algorithm case is that it changes the summation to maximization in the case of Viterbi.

4.3.3 Baum-Welch Re-estimation

Problem 3 concentrates on “changing” the model to fit the observed sequence. This can be done in various ways, including gradient descent. Baum-Welch is a standard method that is used for tuning a given model; it calculates the frequency counts of each transition and emission probabilities of a given model using forward and backward scores.

Backward algorithm is used to calculate the backward score of the observed sequence. It is similar to the forward algorithm except that it traces the given sequence from the back (i.e. considering the last symbol of the sequence emitted by the last match or insert state.)

Backward Algorithm, in the case of a Profile Hidden Markov Model, is [1]:

$$B_k^M(i) = \log \left[\begin{array}{l} a_{M_k M_{k+1}} e_{M_{k+1}}(x_{i+1}) \exp(B_{k+1}^M(i+1)) \\ + a_{M_k M_k} e_{M_{k+1}}(x_{i+1}) \exp(B_{k+1}^I(i+1)) \\ + a_{M_k D_{k+1}} \exp(B_{k+1}^D(i)) \end{array} \right]$$

$$B_k^M(i) = \log \left[\begin{array}{l} a_{I_k M_{k+1}} e_{M_{k+1}}(x_{i+1}) \exp(B_{k+1}^M(i+1)) \\ + a_{I_k I_k} e_{I_k}(x_{i+1}) \exp(B_k^I(i+1)) \\ + a_{I_k D_{k+1}} \exp(B_{k+1}^D(i)) \end{array} \right]$$

$$B_k^D(i) = \log \left[\begin{array}{l} a_{D_k M_{k+1}} e_{M_{k+1}}(x_{i+1}) \exp(B_{k+1}^M(i+1)) \\ + a_{D_k I_k} e_{I_k}(x_{i+1}) \exp(B_k^I(i+1)) \\ + a_{D_k D_{k+1}} \exp(B_{k+1}^D(i)) \end{array} \right]$$

The base case for Backward algorithm :

$$B_{M_{M+1}}(L+1) = 0$$
$$B_{M_M}(L) = \log(a_{M_M M_{M+1}})$$
$$B_{I_M}(L) = \log(a_{I_M M_{M+1}})$$
$$B_{D_M}(L) = \log(a_{D_M M_{M+1}})$$

Baum-Welch is a special case of the Expectation Maximization algorithm that tunes existing transition and emission probabilities depending upon how often each one of them is used (a detailed discussion of it can be found in [1] and [4]). Baum-Welch re-estimation equations in the case of Profile Hidden Markov Models are [1]:

Expected emission counts from sequence x:

$$E_{M_k}(a) = \frac{1}{P(x)} \sum_{i|x_i=a} f_{M_k}(i) b_{M_k}(i)$$

$$E_{I_k}(a) = \frac{1}{P(x)} \sum_{i|x_i=a} f_{I_k}(i) b_{I_k}(i)$$

Expected transition counts from sequence x:

$$A_{X_k M_{k+1}} = \frac{1}{P(x)} \sum_i f_{X_k}(i) a_{X_k M_{k+1}} e_{M_{k+1}}(x_{i+1}) b_{M_{k+1}}(i+1)$$

$$A_{X_k I_k} = \frac{1}{P(x)} \sum_i f_{X_k}(i) a_{X_k I_k} e_{I_k}(x_{i+1}) b_{I_k}(i+1)$$

$$A_{X_k D_{k+1}} = \frac{1}{P(x)} \sum_i f_{X_k}(i) a_{X_k D_{k+1}} b_{D_{k+1}}(i)$$

In the above equations f and b represent the forward and backward scores respectively. The emission and transition scores calculated from the above sequences are iterated until a stop criterion is reached. The stop criterion is generally the maximum number of iterations or the change in the scores is less than a predefined value [1].

5. ANTIVIRUS TECHNOLOGIES

The war between viruses and antivirus(AV) technologies has continued for more than a decade now. VXHeaven alone has a collection of about 66,000 malicious code constructs, but not all of these viruses are out in the wild. Organizations like “The WildList Organization International” release a monthly list of viruses that are most likely to attack. WildList [15] is a collection of viruses known to be spreading in the wild that are confirmed by researchers all over the world. Some AV technologies test their product against these viruses before they are released. AV suppliers constantly work in detection and restoration processes, but are surreptitious about their new methods. The following sections contain a brief description about the most popular methods used to detect viruses today.

5.1 Signature Scanners

Signature Detection is the oldest and most popular virus detection technique used today. Each virus is searched for a string of bytes that is unique to it, which becomes the signature of the virus. Signatures, also called “Scan Strings,” sometimes depend upon the placement in the virus code. Scanners use a signature collection to identify known viruses and are almost certain to detect them. By constantly increasing virus’s collection, signature scanning should be effective and efficient.

A constant string is easy to find, but today’s viruses use obfuscation to escape string scanning. Signatures need to be tweaked to catch these different stains. Reordering of code is a simple method used to cheat scanners. To detect these differences, scanners consider the code a match even if it has a different byte order than the signature.

Signatures can also contain wild cards that allow a few bytes to be anything. In the case of register swapping, the signature differs by the few bytes that contain the registers, but the other bytes remain same. In such cases wild cards are beneficial in identifying new strains with an old signature. Signature extraction is a challenge in itself; a small signature would match to other non-virus programs and a long signature would be over fitting and may not identify new strains. To overcome this, multiple scanners are used on the same system. These scanners use a different set of signatures and help in identifying whatever signatures one scanner misses.

Scanners can be proactive or reactive [16]; proactive scanners continuously scan the access files, whereas reactive scanners are on-demand scanners and work as scheduled. Proactive scanning affects the performance of the system but is very efficient in handling the virus threats as soon as possible. On the other hand, reactive scanners will not affect the performance but might not detect the virus until it is too late. Whichever scanner is used, it has to be updated as there are new signatures made available by their vendors. Vendors like AVG supply free downloads of antivirus toolkits for home users, which update automatically every day. Although scanners are not slow these days, emerging new viruses can add up and affect their performance. Different AV vendors deal with them differently; some of them take into consideration the type of file being scanned, and that gives them a hint of what part of the code they should look at.

As discussed in section 2, viruses are clever at changing their look with alternating source code. A good mutation engine will generate very different strains and each strain will not have the signature of the original virus. In the case of polymorphic and metamorphic viruses, it is not possible to have a unique signature for the virus family. This means that although signatures of various strains are known there is always a good chance that another strain will succeed in bypassing the signature detection.

5.2 Checksum

Checksum is used to verify the integrity of any kind of files. It is normally used to check the correctness of TCP/IP packets that are the main source of communication on the Internet. Software manufacturers use checksum to detect unauthorized modifications made to bypass their license check. The concept of checksum is also used in generating message authentication code (MAC) to check the integrity of messages [6]. Today’s viruses also use checksum to see if their code is tampered with before it starts infecting.

There are many checksum programs that are readily available for download. Since they are called only when a new program is accessed, they do not have a high performance impact. Executable files are not changed often, so a checksum can be used to verify their integrity. When an integrity check fails, there is a chance that a virus will have modified it and this helps in detecting the malicious behavior. Checksum is an example of “detection by change” methodology, where a malicious activity is detected when files are changed.

Checksum is a traditional method of detecting the unwanted changes; however, there are a few viruses like the latest Hidan [17] from the Chiton family of W32 viruses that will calculate a new checksum after infection. It later replaces the existing checksum with the new value, thus escaping the detection.

5.3 Hardware-based security

Next Generation Secure Computing Base (NGSCB) is a hardware-based security system that allows only “trusted” agents to access secrets on the system. These secrets can be memory, signatures and keys used by the user. Unlike other AV tools these systems need not depend on a particular virus and have common detection mechanisms for all malware. However, an operating system needs to be configured in order to use this system.

Apart from using NGSCB to sign documents, digital rights management [6] can be used to keep viruses at bay. Access control lists (ACL) are often used in an authorization process, and are checked to see if a user is allowed to perform an action. Viruses will never be given access to perform malicious activities if ACLs for each application are maintained properly. In other words a proper authorization for applications is needed in a system where privilege for each application is clearly defined.

The operating system has to be configured to use this system. As it can also be programmed to identify if an application is behaving oddly, this can be taken as an anti-virus technology. Efficiency of this system depends upon how frequently new applications are used. A home user might need to rebuild the complete access matrix every time new software is installed and this imposes considerable overhead [16]. On the other hand, at an organizational level which does not change often, this would be a very good solution. An experienced system administrator would know which applications are allowed to do what.

The toughest problem in this system is how to measure the trustworthiness of an application. To set the allowed operations of an applications, definitions of what is not malicious need to be defined, which again depends upon what existing malware has caused or might cause. There is always a possibility that viruses will modify or delete these access lists, but then again this is a common problem for all anti-virus products.

5.4 Heuristics Based Analysis

Heuristics is prominently used for discovering unknown viruses depending upon known virus behavior. Every new file is monitored and scored against a predefined set of indicators that are determined through analyzing known viruses. When the score of these indicators is high it is flagged as a virus. Although there are known to be false positives in this process, it is fairly effective in detecting unknown and new strains of viruses.

Static heuristic analysis deals with inspecting code sequences for known virus-like code. A flagged malicious behavior in the static case would trigger the dynamic heuristics. Dynamic heuristics emulate the program under consideration to further explore it. It looks for indicators like very big files, large debug sections, entry-point code redirection, suspicious kernel operation and many more. If the program fails the heuristics test, the user is warned about the same; otherwise the heuristics scanner continues closely watching the program’s system calls and interrupts [23]. Indicators used in the analysis sometimes number in the hundreds. Using too many indicators is disadvantageous as it flags non-viruses, and tweaking the right score threshold poses considerable challenges in using heuristics.

In the case of polymorphic viruses, the code is executed in an emulator until it is decrypted and a known signature is seen; this process needs to be continued in case of multi-layered encryptions. Metamorphic viruses do not have a signature and their detection depends upon the indicators for any doubtful actions. But metamorphic viruses often carry a payload that triggers the virus behavior under certain conditions; in such cases heuristics analysis is cheated. Heuristic analysis is also known to be implemented using neural networks that are as efficient as its training set [11].

5.5 Virtual Machine Execution

Mutation engines used in few viruses use the memory stack for generating variants. Such viruses contain the signatures in the stack and not in the actual code. To detect such viruses, anti-virus researchers should pay attention at the system’s internal working. It is extremely important to execute these viruses in a safe environment so that they do not escape into the wild.

Viruses that are polymorphic contain encrypted code and a virtual machine can be used to step through the instructions until a signature in its decrypted code is detected. Since the virtual machine has all the memory traces and API calls used by the virus it is easier to analyze for any suspicious activities like too many jumps, nop and XOR/NOR instructions. It is helpful in detecting metamorphic strains that use encryption and obfuscations like junk code insertion and code reordering.

Few viruses are intelligent enough to detect a virtual machine and go in to a recursive loop or execute unwanted instructions or exit without executions. Such conditions can be fine-tuned within the machine to alert the user. Code emulation on a virtual machine comes to the rescue when no other methods are helpful, and anti-virus researches use these to debug and analyze new viruses. But in today's world where performance is key, virtual machines are slower and need more resources than any other method.

6. IMPLEMENTATION

For a given multiple sequence alignment (MSA) of opcodes, the goal is to generate a profile hidden markov model and score sequences of both viruses and non-viruses using the model.

A PHMM model is trained depending upon an MSA generated using opcodes sequences from virus files. These virus opcodes used for our project are generated using 3 virus construction kits: Virus creation laboratory (VCL), Phalcon/Skism Mass-Produced Code Generator (PS-MPC) and Next generation virus creation kit (NGVCK) (more detailed description of how these virus kits work is given in section 2.3). Each of these kits is used to generate various variants and grouped under a family. We wanted to test the performance of PHMM over various construction kits that are from different time periods as this will give us a better understanding of the improvements and trends followed by the virus writers.

A PHMM model is a combination of Emission and Transition probabilities per state and per opcode basis. The number of entries of these probabilities depends upon the gaps and symbols in a given MSA. Basically, the model is as strong as the given MSA. A weak MSA with many gaps will result in a model containing few states.

Forward Algorithm is used to score ASM files against a PHMM model. For this purpose, we have used non-virus files from genuine programs normally seen on many systems. These files are filtered to contain only opcodes before they are scored, as any other information like subroutine markers and registers are changed often.

6.1 Test Data Generation and Filtration

Using 3 different construction kits we generated different variants by changing the configuration settings provided by each kit.

Our test data contains:

- 10 variants from VCL (vcl32_01 to vcl32_10)
- 30 variants from PS-MPC (psmpc_01 to psmpc_30)
- 200 different variants from NGVCK (ngvck_001 to ngvcl_200)
- 40 disassembled cygwin dll's of version 1.5.19 (cygwin_01 to cygwin_40)
- 30 disassembled dll's from other non viruses like Msoffice, Adobe, IE… etc (nonvirus_01 to non_virus_30)

These construction kits are downloaded from VXHeaven. There are several versions of each of the kits available and we have used the latest and most stable version for our test data generation.

Table 4 contains the release date and versions of each of the kits used:

Name of the Kit	Version Used	Release Date
PS-MPC	PS-MPC 0.91	August 1992
NGVCK	NGVCK0.30	June 2001
VCL32	VCL32	February 2004

Table 5: Construction kits information

VCL, PS-MPC and NGVCK all produce asm files depending upon their settings and configurations. We have chosen to incorporate the most significant variants in our test data. Although PS-MPC is capable of generating thousands of variants with different payloads, we used the most important configurations like memory resident, encryption, file type, etcetera, to generate the variants. Similarly, with VCL and NGVCK, test data is generated to have at least one of the various settings possible; this will enable us to have our model tuned to expect different variants.

We used IDA Pro Disassembler to disassemble the dll's and exe's of cygwin and other non-viruses. To maintain consistency in the opcodes we wanted to use IDA Pro for disassembling the virus variants too. Since the output of the kits was already in the asm format, we used Turbo Assembler (tasm 5.0) for compiling and linking the files to generate exe's, which are later disassembled using IDA pro.

Virtual machine using VMWare Workstation was used for all virus files processing to keep the viruses in a closed system and all the engines and exe's were deleted after we had the asm file source.

Since each group of viruses is from a different construction kit, they are very different in terms of the opcodes used. All three construction kits used generate 32-bit PE executable files and each of these files can contain any of the 250 x 86 opcodes. Using all of these different opcodes would make the emission and transition probabilities too small; besides there are only 14 opcodes that are most likely to be seen in malware as well as genuine programs [24]. Depending upon opcode frequencies in virus variants, we generated one alphabet for each virus family containing 37 different opcodes.

A wild character "*" is used for any opcodes that are not in the top 36 opcodes and this is essential, as any opcode might show up during scoring. The alphabet thus generated is fixed and used throughout the process for MSA, modeling and scoring of the virus family.

In the models we generated the probabilities perceived for "*" are much less than the other opcodes; thus, a sequence not belonging to the virus family will have different opcodes and a higher chance of using the lower probability opcodes.

Each asm source file of the viruses is filtered to contain only the opcodes, while other information like the subroutine names, registers and comments are omitted. This takes care of the early metamorphic viruses like Regswap that used only register swapping. These filtered files are now used to generate the MSA and Scoring.

6.2 Training the Model

The multiple sequence alignment we used as an input to our modeling algorithm is generated using the Feng-Doolittle progressive multiple alignment algorithm [25]. A PHMM model created from observing the MSA of its variants carries data about opcodes patterns for each virus family. We have followed a general method used for training the model as explained in section 3.2.1.

A model can be generated for each virus family containing all the virus variants generated, or a model can be generated for each of the subgroups of the variants. But we opted to generate more than one model for each virus family, giving us the flexibility to test our method against other virus variants of the same family.

After looking at various MSA's generated by grouping a variable number of files we decided to group them as follows:

VCL32 – 2 groups with 5 files in each group

PS-MPC – 3 groups with 10 files in each group

NGVCK – 10 groups with 20 files in each group

The percentage of gaps perceived in the virus families is shown in Table 6. These gap percentages give us a raw estimation of the PHMM model performance. An MSA with many gaps is more generic and might lose the virus-specific information, especially in advanced metamorphic cases.

Virus Family	**Gap %**
VCL32	7.453
PS-MPC	23.555
NGVCK	88.308

Table 6: Gap percentages perceived in MSA's of each Virus family

As it can be seen NGVCK generates far more diverse variants than other construction kits.

The following are the steps used for training the model:

- Calculate the begin probabilities. These are the transition probabilities from the begin state to the first insert, match and delete states. In our case, we have measured the begin state to be another match state and renamed it as $M_{0,}$ which will enable us to use the recursive forward algorithm efficiently.
- Identify the match states. We used MSA columns with more than half filled as match and the rest as insert. In the case of Bioinformatics, an experienced biologist would determine this.
- Calculate the emission probabilities. Each MSA is considered as a group of columns with symbols of opcodes in them; each of these columns is traversed and frequencies of each opcode is noted. These frequencies are later used to calculate emission match and emission insert probabilities.

- Calculate the transition probabilities. Each column is traversed to store the number of transitions between each of the match, insert and delete states. These results are used in calculating the final transition probabilities perceived in the alignment.
- Calculate the end probabilities. The last match state is the end state, if there are n states we renamed our match state as $M_{(n+1)}$. Since begin and end match states are the only match states that do not emit any symbols, there are no emission probabilities pertaining to them.

The model generated for VCL32 group 1 using files numbered vcl32_01 to vcl32_05 contains a total of 1820 states with emission probabilities and transition probabilities. Table 6 shows emission probabilities seen for states 126, 127 and 128 calculated from a multiple sequence alignment of 5 files (vcl32_01 to vcl32_05):

	Emission Match Probabilities			Emission Insert Probabilities		
opcodes	State 126	State 127	State 128	State 126	State 127	State 128
and	0.0238	0.025	0.025	0.0612	0.0256	0.0256
inc	0.0238	0.025	0.025	0.0204	0.0256	0.0256
xor	0.0238	0.025	0.025	0.0204	0.0256	**0.0513**
stc	0.0238	0.025	0.025	0.0204	0.0256	0.0256
stosb	0.0238	0.025	0.025	0.0204	0.0256	0.0256
imul	0.0238	0.025	0.025	0.0204	0.0256	0.0256
jecxz	0.0238	0.025	0.025	0.0204	0.0256	0.0256
jmp	0.0238	0.025	0.025	0.0204	0.0256	0.0256
shl	0.0238	0.025	0.025	0.0204	0.0256	0.0256
not	0.0238	0.025	0.025	0.0204	0.0256	0.0256
add	0.0238	**0.1**	0.025	0.0612	0.0256	0.0256
stosd	0.0238	0.025	0.025	0.0204	0.0256	0.0256
call	0.0238	0.025	0.025	0.0612	0.0256	0.0256
jnz	0.0238	0.025	0.025	0.0204	0.0256	0.0256
push	0.0238	0.025	0.025	0.0204	**0.0769**	**0.0513**
cmp	0.0238	0.025	0.025	0.0204	0.0256	0.0256
dec	0.0238	0.025	0.025	0.0204	0.0256	0.0256
xchg	0.0238	0.025	0.025	0.0204	0.0256	0.0256
test	0.0238	0.025	0.025	0.0204	0.0256	0.0256
*	0.0238	0.025	0.025	0.0204	0.0256	0.0256
jb	0.0238	0.025	0.025	0.0204	0.0256	0.0256
sub	0.0238	0.025	0.025	0.0612	0.0256	0.0256
or	0.0238	0.025	0.025	0.0204	0.0256	0.0256
jz	0.0238	0.025	0.025	0.0204	0.0256	0.0256
neg	0.0238	0.025	0.025	0.0204	0.0256	0.0256
retn	0.0238	0.025	0.025	0.0204	0.0256	0.0256

lodsb	0.0238	0.025	0.025	0.0204	0.0256	0.0256
mov	**0.1429**	0.025	**0.1**	**0.102**	0.0256	0.0256
pop	0.0238	0.025	0.025	0.0204	0.0256	0.0256
jnb	0.0238	0.025	0.025	0.0204	0.0256	0.0256
shr	0.0238	0.025	0.025	0.0204	0.0256	0.0256
stosw	0.0238	0.025	0.025	0.0204	0.0256	0.0256
lodsd	0.0238	0.025	0.025	0.0204	0.0256	0.0256
cld	0.0238	0.025	0.025	0.0204	0.0256	0.0256
rep	0.0238	0.025	0.025	0.0204	0.0256	0.0256
lea	0.0238	0.025	0.025	0.0204	0.0256	0.0256
rol	0.0238	0.025	0.025	0.0204	0.0256	0.0256

Table 7: Emission Match and Insert Probabilities for VCL32 Group1 in States 126, 127 and 128

As can be seen, there are few opcodes that occur more often than other opcodes. The add-one rule is used for opcodes that are not seen at all instead of using a zero probability, which enables us to accommodate them in scoring instead of ignoring them.

The transition probabilities between states 126, 127 and 128 for group1 VCL32 files are given below:

	$\mathbf{M_{127}}$	$\mathbf{I_{127}}$	$\mathbf{D_{127}}$		$\mathbf{M_{128}}$	$\mathbf{I_{128}}$	$\mathbf{D_{128}}$
$\mathbf{M_{126}}$	0.500	0.375	0.125	$\mathbf{M_{127}}$	0.667	0.167	0.167
$\mathbf{I_{126}}$	0.067	0.733	0.200	$\mathbf{I_{127}}$	0.200	0.200	0.600
$\mathbf{D_{126}}$	0.333	0.333	0.333	$\mathbf{D_{127}}$	0.200	0.600	0.200

Table 8: Transition probabilities between states 149,150 and 151 for group1 NGVCK

The probabilities shown in Table 8 can be interpreted as:

$a_{M_{126}M_{127}}$ = 0.5, probability that M_{127} is reached after M_{126} emits a symbol is greater than I_{127} and D_{127} are reached. Notice that the sum of the probabilities of each row is equal to 1 and so is the sum of each column in the emission probabilities.

The time complexity of the method used to implement PHMM training is O(nL), where n is the number of sequences in the MSA and L is the length of training sequence.

6.3 Forward Scoring

Forward algorithm scores a given sequence against a given HMM model using the principles of dynamic programming. It is a recursive procedure that reuses the scores generated in its previous steps. The theory and formulas used for our project are stated in section 3.3.1.

The following are the steps involved in scoring:

- To score a given sequence X (x_1, x_2,….x_L) against a PHMM with N+1 states(0,1…N) with N >= 1, states 0 and N being the start and end states respectively, we proceed by calculating $F^M_{N-1}(L)$, $F^I_{N-1}(L)$ and , $F^D_{N-1}(L)$ in that order.

- In the recursive process of calculating $F^M_{N-1}(L)$, many other intermediate values like $F^M_{N-2}(L-1)$, $F^I_{N-1}(L-1)$….. are calculated and stored for later use. By the time $F^D_{N-1}(L)$ is calculated, very few intermediate scores have to be calculated from scratch, thus making scoring efficient.

  Figure 11 explains this recursion process:

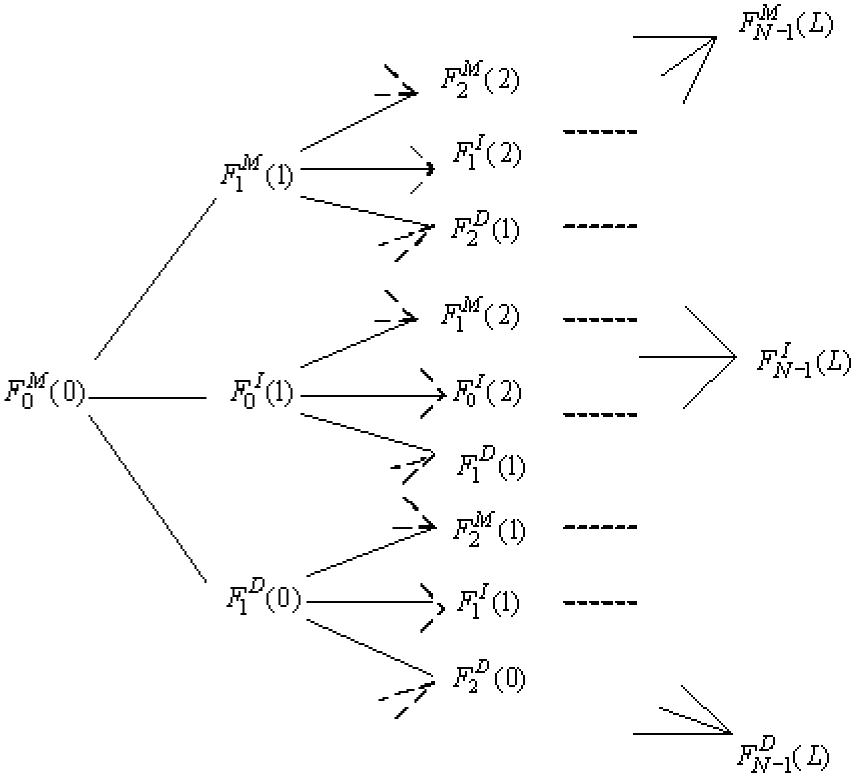


Figure 11: Forward Algorithm recursive approach

- During the calculations there are a few terms like $F^I_0(0)$, $F^M_0(2)$, … which are not defined; when these are encountered, we simply exclude them from the calculations.

- $F^M_{N-1}(L)$, $F^I_{N-1}(L)$ and $F^D_{N-1}(L)$ represent the scores for sequence X until it reaches the N-1 states; multiplying these scores with their respective end transition probabilities gives the final score.

$$\begin{tikzcd}
F^{M}_{N-1}(L) \arrow[dr, "a_{M_{N-1}M_N}"] & \\
F^{I}_{N-1}(L) \arrow[r, "a_{I_{N-1}M_N}"] & F^{M}_{N}(L) \\
F^{D}_{N-1}(L) \arrow[ur, "a_{D_{N-1}M_N}"'] &
\end{tikzcd}$$

Figure 12 Final Score from previous states

$$\text{Total Score} = \log\left[\begin{array}{c} a_{M_{N-1}M_N}\,\exp(F^{M}_{N-1}(L)) + a_{I_{N-1}M_N}\,\exp(F^{I}_{N-1}(L)) \\ +\,a_{D_{N-1}M_N}\,\exp(F^{D}_{N-1}(L)) \end{array}\right]$$

- The scores thus generated are log-odds scores and hence we don't have to subtract any random or null model scores as normally done in HMM scoring.

The resultant scores are sequence-length dependant and cannot be used directly for comparison. We divided the final score by sequence length, giving us per-opcode basis scores. Since all the scores are per-opcode, these can now be used to directly compare with other scores.

Due to the logarithms used in the scoring process, we did not have any underflow problems, but due to the exponentiation part of the calculation, there were overflow problems. The intermediate score sometimes reached greater than 700 and exp(700) is a very large number, which affects performance. To overcome these problems, we used the following mathematical principle mentioned in [1]:

$$\log(p+q) = \log(p) + \log(1+\exp(\log(q)-\log(p))$$

Exponentiation of a big number is not necessary as they are changed as the difference between logarithmic values. In special cases where there is only one term, log and exp cancel each other out.

Since there are a fixed number of possible transitions from each state, the time complexity is O(nT) where n is the number of states and T is the length of the observed sequence.

7. RESULTS

The score of a given sequence using a virus family model represents its similarity to the virus. High scored sequences are more closely related to the virus whereas lower scored sequences are more diverged and thus are less probable to be viruses.

We have scored non-viruses and virus variants of each construction kit against various PHMM models representing the virus family. Test data grouping and model names are shown in Table 9.

Virus Family	Groups/Model Name	Files in Group
VCL32	vcl32_group5_1	vcl32_01 to vcl32_05
	vcl32_group5_2	vcl32_06 to vcl32_10
PS-MPC	psmpc_group10_1	psmpc_01 to psmpc_10
	psmpc_group10_2	psmpc_11 to psmpc_20
	psmpc_group10_3	psmpc_21 to psmpc_30
NGVCK	ngvck_group20_01	ngvck_01 to ngvck_020
	ngvck_group20_02	ngvck_021 to ngvck_040
	ngvck_group20_03	ngvck_041 to ngvck_060
	ngvck_group20_04	ngvck_061 to ngvck_080
	ngvck_group20_05	ngvck_081 to ngvck_100
	ngvck_group20_06	ngvck_101 to ngvck_120
	ngvck_group20_07	ngvck_121 to ngvck_140
	ngvck_group20_08	ngvck_141 to ngvck_160
	ngvck_group20_09	ngvck_161 to ngvck_180
	ngvck_group20_10	ngvck_181 to ngvck_200

Table 9: Test Data Grouping and Model Names

The default threshold for log-odd scores is 0, that is, log-odd scores would be positive for family variants and negative for non-family members. A positive threshold greater than zero can also be used but carries a risk of detecting non-family files as viruses, and vice versa.

Since we have used diverse variants while modeling each virus family and have a considerable dataset of known to be viruses, the threshold is taken as the minimum score from the viruses of each family.

Figure 13 shows the scatter plot of scores against the vcl32_group5_1 model.

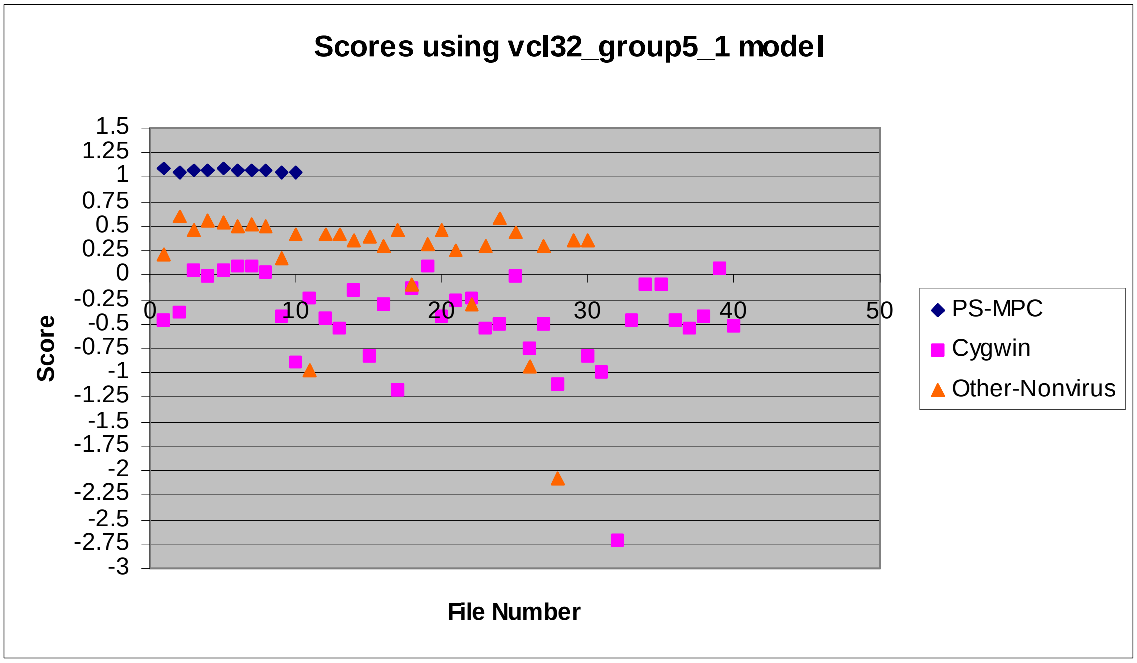


Figure 13 Scores for Virus and Non Virus files using vcl32_group5_1 model

There are no scores from the cygwin or other non-viruses that are greater than the minimum score of 1.0546 in vcl32 variants, thus clearly distinguishing non-viruses from vcl family viruses. Scores against the models of vcl are included in Appendix (Table A-1 and Table A-2).

Results for psmpc_group10_1 are shown in Figure 14. There are no false positives or false negatives using all three models generated from PS-MPC. Thus the detection rate perceived in VCL32 and PS-MPC is 100% with a false positive rate of 0%.

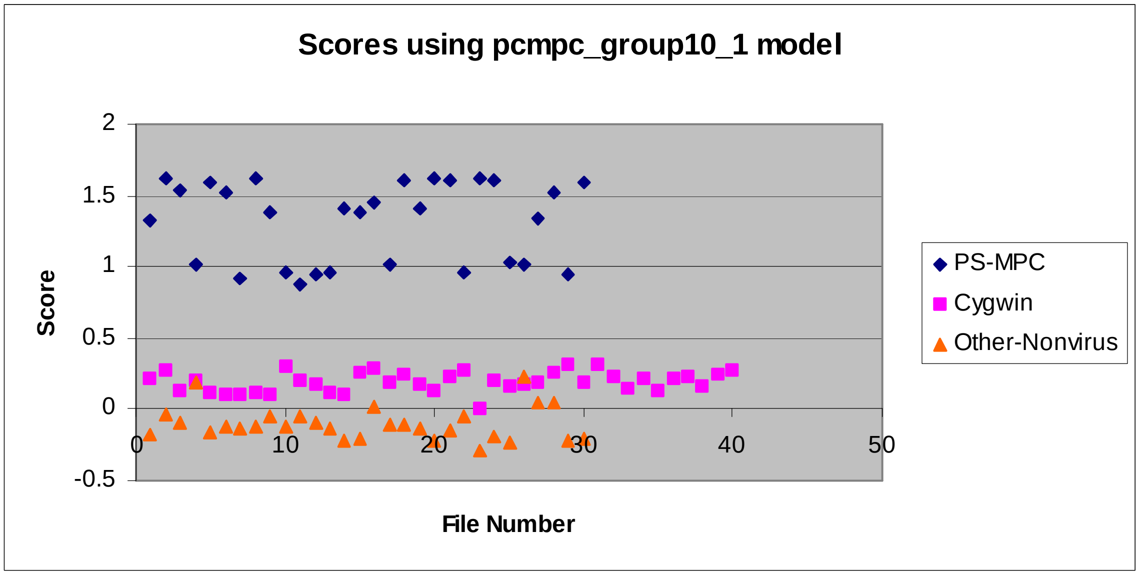


Figure 14 Scores for Virus and Non Virus files using psmpc_group10_1 model

NGVCK, as seen from the gap percentages (Table 6), is more advanced than PS-MPC and VCL32. Figure 15 shows the results using the ngvcl_group20_01 model. Non-virus files that score greater than 0.715 are considered false positives.

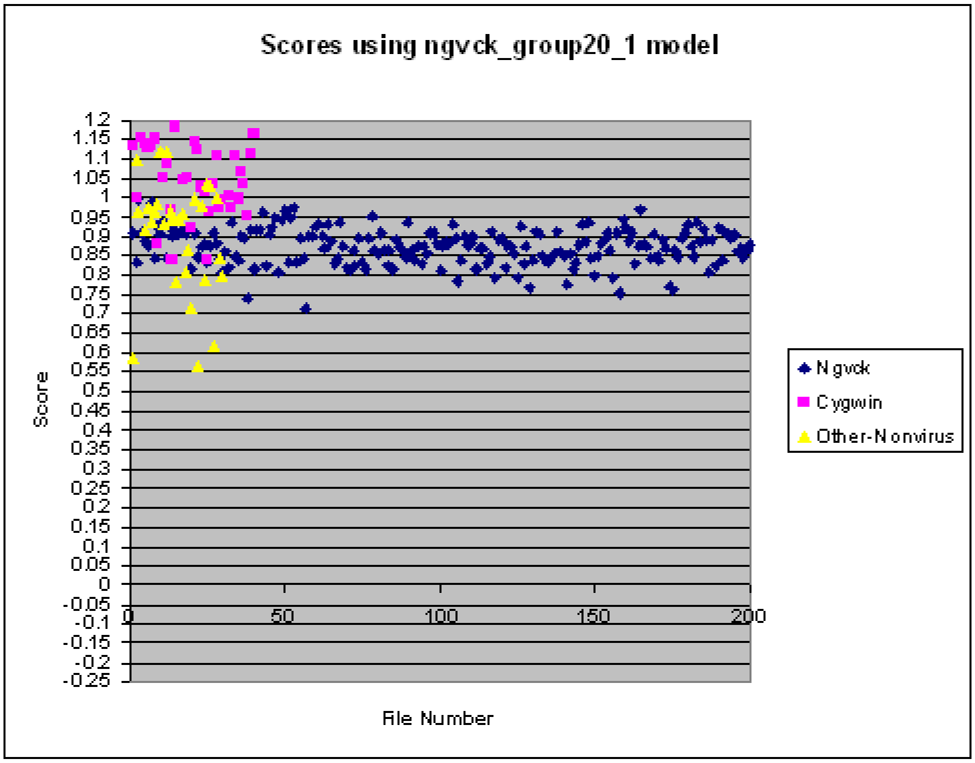


Figure 15 Scores for Virus and Non Virus files using ngvck_group20_01 model

The increased rate of false-positives in the NGVCK case is due to the subroutine permutation used by the construction kit. As different variants had different subroutine order, the opcodes in the MSA are not aligned as intended. For example, consider assembly files file1.asm and file2.asm with 3 subroutines each, where the order of subroutines in file 1 is (1,2,3) and (2,3,1) in case of file 2. The MSA generated from these files has aligned subroutine 1 in file 1 with subroutine 2 in file2, giving considerable gaps in the final MSA.

To overcome this problem, we generated new models for NGVCK viruses using finely-tuned MSA's. New set of MSA's created for this purpose used virus files that are reordered to contain fewer gaps. (More details about the preprocessing can be found in [25]). We will be referring to these files as preprocessed files from now on. The MSA gap percentage of NGVCK variants decreased from 88.3% to 44.9% percent using the preprocessed files. In a real-world scenario the source file we score can be a virus or a non-virus, so a preprocessing step is essential for any file to be scored. The models generated from preprocessed files are named as ngvck_pp_group20_01. The virus and non-virus files used for scoring are from now on are all preprocessed.

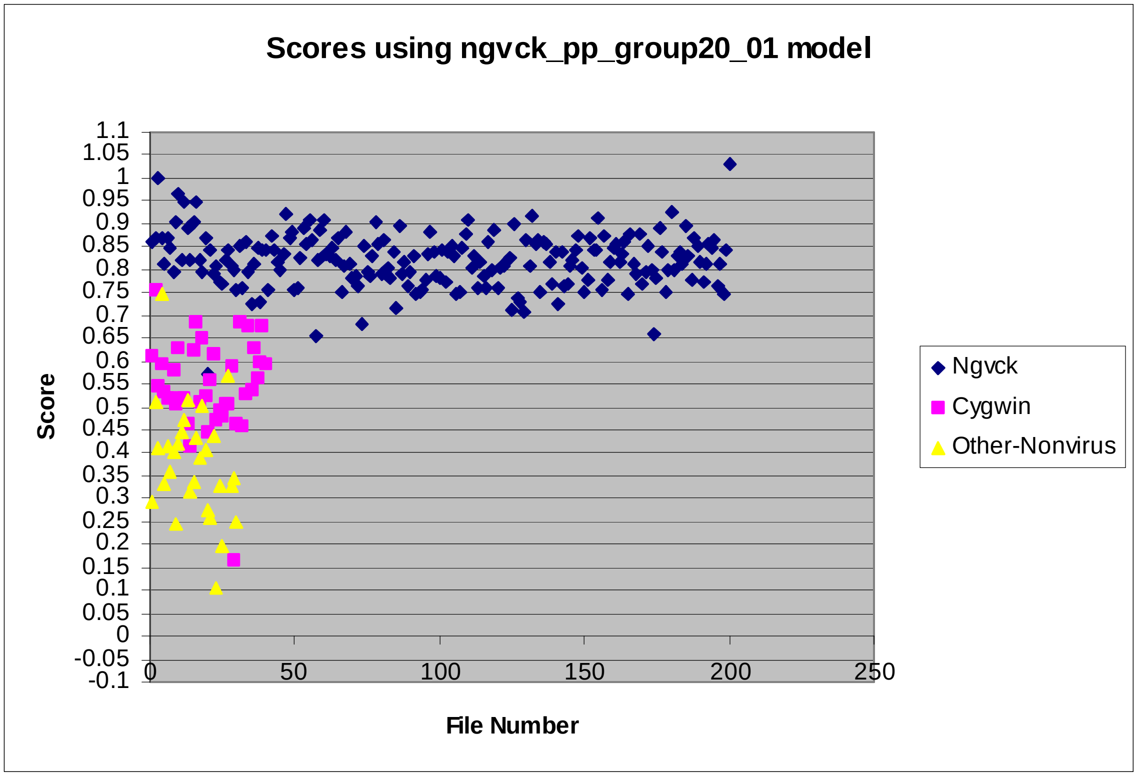


Figure 16 Scores for Virus and Non Virus files using ngvck_pp_group20_01 model

Figure 16 shows the scores for preprocessed files using the ngvck_pp_group20_01 model. Although the false positives are not completely gone, the average false-positive rate across all NGVCK models decreased from 92.57% to 48.43% and the overall accuracy has considerably increased from 75.93% to 95.92% with the preprocessing step.

Since few virus variants scored much less than the other files, we increased the threshold to second and third minimum scores perceived in the virus variants. Increasing the threshold would allow actual virus files to bypass the detection, increasing false negatives. The average false-negative rate over all groups of NGVCK pre-processed files was 1% in the case of third minimum threshold and 0.5% in the case of second minimum threshold.

The improvement due to pre-processing the files can be seen clearly by calculating the false-positive percentages before and after the pre-processing step at various threshold levels.

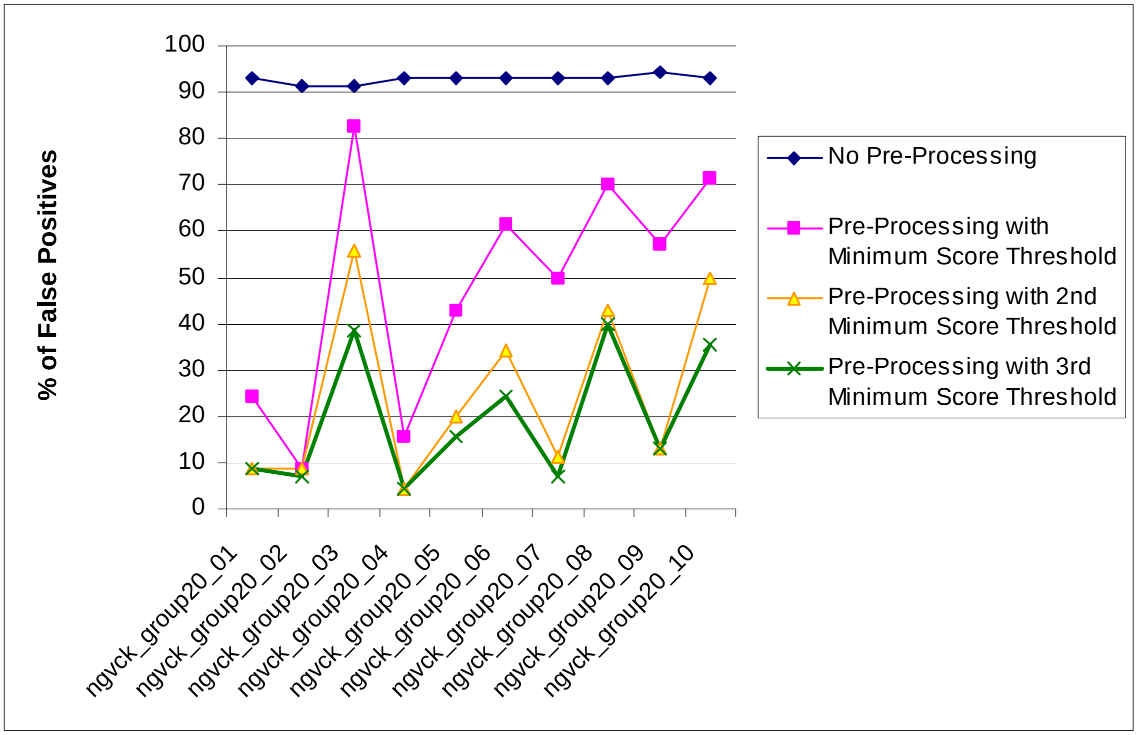


Figure 17: False Positive Percentages for Non-virus Before and After Preprocessing at Different Thresholds

As shown in Figure 17, the number of false positives decreased considerably by increasing the threshold and preprocessing the files. Since increasing the threshold to third minimum of the virus scores has improved the accuracy rate, with a good balance of false positives and false negatives, we can use a third minimum threshold for the NGVCK viruses. Due to space constraints, we have added scores calculated using all NGVCK models in Appendix C. Although the accuracy is not 100% as in the case of VCL32 and PS-MPC, NGVCK viruses can be detected with few false positives and false negatives.

8. CONCLUSION

Virus detection is crucial in today's world of computers. Metamorphic viruses are far more advanced and harder to detect than any other kind of viruses in the wild. In this report, we have described the challenges most anti-virus technologies face in detecting metamorphic viruses.

Profile Hidden Markov Models (PHMM) are known for their success in determining relations between DNA and protein sequences. We have experimented to see whether PHMM can be used in detecting computer virus variants generated using construction kits. Our results show that Profile Hidden Markov Models can be successfully used to model viruses. Using a faster approach called Forward algorithm, we calculated the scores for virus and non-virus files (like cygwin dll's and application dll's) against each virus model. The time complexity to score using a PHMM is O(nT), where n is the number of states and T is the length of the sequence.

We tested our method on three construction kits--namely VCL32, PS-MPC and NGVCK, which use simple to advanced code-morphing techniques. The results showed a 100% detection with 0% false positive and false negative rates in VCL32 and PS-MPC. After rearranging the subroutines and threshold tuning, we were able to detect NGVCK viruses with a false positive rate of 19.43% and a false negative rate of 1%.

The relationship between opcodes sequences in virus family variants and non-viruses is different and PHMM can model that accurately. Detecting metamorphic viruses using Profile Hidden Markov Models is highly feasible, based on performance and results.

9. FUTURE WORK

The following ideas can be used to further extend the concept of PHMM in detecting metamorphic viruses:

- Our test data contains variants of three construction kits. When other variants of the same virus families are discovered, a new set of models that include the newly-detected variants needs to be generated using our method. One alternative, would be to tune the emission and transition probabilities in the PHMM model using the Baum-Welch reestimation method.
- We have trained our models using assembly sources of virus files. This can be extended to model each subroutine and calculate an aggregate score. Subroutine modeling might detect metamorphic viruses that implement subroutine permutation and code reordering. On the other hand, more advanced obfuscations that generate different subroutines for their variants would be a greater challenge to detect.
- Training and scoring are faster than heuristics-based techniques, but the time taken to filter the data, and the disassembling, can hinder the performance of different kinds of files. It would be interesting to see how PHMM performs if binary code is used directly.

REFERENCES

[1] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, “Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids,” Cambridge University Press, 1988.

[2] A. Krogh, “An Introduction to Hidden Markov Models for Biological Sequences,” Center for Biological Sequence Analysis, Technical University of Denmark, 1988.

[3] D.W. Mount, “Bioinformatics: Sequence and Genome Analysis,” Cold Spring Harbor Laboratory, 2004.

[4] L.R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” Proceedings of the IEEE, Volume 77, Issue 2, Feb. 1989. Pages 257-286.

[5] M. Stamp, “A Revealing Introduction to Hidden Markov Models,” January 2004.

http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf.

[6] M. Stamp, “Information Security: Principles and Practice,” August 2005.

[7] P. Szor, P. Ferrie, “Hunting for Metamorphic,” Symantec Security Response.

http://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf.

[8] S.R. Eddy, “Profile Hidden Markov Models,” Bioinformatics, Oxford Journals, Volume 14, Number 9, July 1998. Pages 755-763.

[9] W. Wong, “Analysis and Detection of Metamorphic Computer Viruses,” Master’s thesis, San Jose State University, 2006.
http://home.earthlink.net/~mstamp1/mss_v.html#masters.

[10] S.Khuri, “Hidden Markov Models,” lecture notes.
http://www.cs.sjsu.edu/faculty/khuri/Bio_CS123B/Markov.pdf.

[11] P.Szor, “The Art of Computer Virus Defense and Research,” Symantec Press, 2005.

[12] R.G. Fiñones and R. Fernandez, “Solving the Metamorphic Puzzle,” Virus Bulletin, Mar. 2006. Pages 14-19.

[13] J. Mc afee and C. Haynes, “Computer Viruses, Worms, Data Diddlers, Killer Programs and Other Threats to Your System,” St. Martin’s Press, 1989.

[14] http://en.wikipedia.org/wiki/Timeline_of_notable_computer_viruses_and_worms.

[15] http://www.wildlist.org/WildList/.

[16] W.T. Polk, L.E. Bassham, J.P. Wack and L.J. Carnahan, ”Anti-virus Tools and Techniques for Computer Systems,” Noyes Data Corporation, 1995.

[17] P. Ferrie, “Hidan and Dangerous,” Virus Bulletin, Mar. 2007. Pages 14-19.

[18] A. Walenstein, R. Mathur, M.R. Chouchane and A. Lakhotia, "Normalizing Metamorphic Malware Using Term Rewriting," Proc. Int'l Workshop on Source Code Analysis and Manipulation (SCAM), IEEE CS Press, Sept. 2006. Pages 75–84.

[19] http://vx.netlux.org/vx.php?id=tp00.

[20] Myles Jordan, “Anti-Virus Research - Dealing with Metamorphism,” Virus Bulletin, Oct. 2002.

[21] http://www.symantec.com/security_response/writeup.jsp?docid=2000-122010-0045-99&tabid=2.

[22] “The Molecular Virology of Lexotan32: Metamorphism Illustrated,” OpenRCE.org, Aug. 2007. http://www.openrce.org/articles/full_view/29.

[23] Jay Munro, "Antivirus Research and Detection Techniques.” ExtremeTech., July 2002. FindArticles.com. 02 Nov. 2007.
http://findarticles.com/p/articles/mi_zdext/is_200207/ai_ziff28916.

[24] D. Bilar, “Statistical Structures: Fingerprinting Malware for Classification and Analysis,” http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Bilar.pdf.

[25] S.McGhee, “Pairwise Alignment of Metamorphic Computer Viruses,” Master’s project, San Jose State University, 2007.
http://www.cs.sjsu.edu/faculty/stamp/students/mcghee_scott.pdf

APPENDIX A - VCL32 Scores

Table A-1 Scores of Virus and Non Virus files using vcl32_group5_1 model

VCL32 Virus Variants		Non Virus Files			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
Vcl32_01	1.083767	Cygwin_01	-0.45906	nonvirus_01	0.209929
Vcl32_02	1.054556	Cygwin_02	-0.37755	nonvirus_02	0.606955
Vcl32_03	1.07452	Cygwin_03	0.044363	nonvirus_03	0.447682
Vcl32_04	1.077914	Cygwin_04	-0.00845	nonvirus_04	0.556673
Vcl32_05	1.094975	Cygwin_05	0.042635	nonvirus_05	0.531772
Vcl32_06	1.067547	Cygwin_06	0.098187	nonvirus_06	0.494801
Vcl32_07	1.069215	Cygwin_07	0.085779	nonvirus_07	0.510706
Vcl32_08	1.080612	Cygwin_08	0.036963	nonvirus_08	0.490268
Vcl32_09	1.060052	Cygwin_09	-0.42124	nonvirus_09	0.179993
Vcl32_10	1.05712	Cygwin_10	-0.89192	nonvirus_10	0.423765
		Cygwin_11	-0.23544	nonvirus_11	-0.98025
		Cygwin_12	-0.43307	nonvirus_12	0.412032
		Cygwin_13	-0.55189	nonvirus_13	0.412032
		Cygwin_14	-0.16056	nonvirus_14	0.357063
		Cygwin_15	-0.83461	nonvirus_15	0.391026
		Cygwin_16	-0.30853	nonvirus_16	0.291146
		Cygwin_17	-1.18801	nonvirus_17	0.461129
		Cygwin_18	-0.13747	nonvirus_18	-0.09653
		Cygwin_19	0.081736	nonvirus_19	0.308743
		Cygwin_20	-0.42498	nonvirus_20	0.454242
		Cygwin_21	-0.25938	nonvirus_21	0.259071
		Cygwin_22	-0.23532	nonvirus_22	-0.29306
		Cygwin_23	-0.54901	nonvirus_23	0.291158
		Cygwin_24	-0.50752	nonvirus_24	0.583751
		Cygwin_25	-0.02293	nonvirus_25	0.443853
		Cygwin_26	-0.75277	nonvirus_26	-0.93934
		Cygwin_27	-0.49897	nonvirus_27	0.300514
		Cygwin_28	-1.11758	nonvirus_28	-2.07051
		Cygwin_29	-6.38913	nonvirus_29	0.350297
		Cygwin_30	-0.83096	nonvirus_30	0.356699
		Cygwin_31	-0.98737		
		Cygwin_32	-2.70584		
		Cygwin_33	-0.45342		
		Cygwin_34	-0.10282		
		Cygwin_35	-0.09447		
		Cygwin_36	-0.45365		
		Cygwin_37	-0.53924		
		Cygwin_38	-0.41534		
		Cygwin_39	0.066167		
		Cygwin_40	-0.52667		

Figure A-1: Graphical representation of Virus and Non-Virus Scores using vcl32_group5_1 model

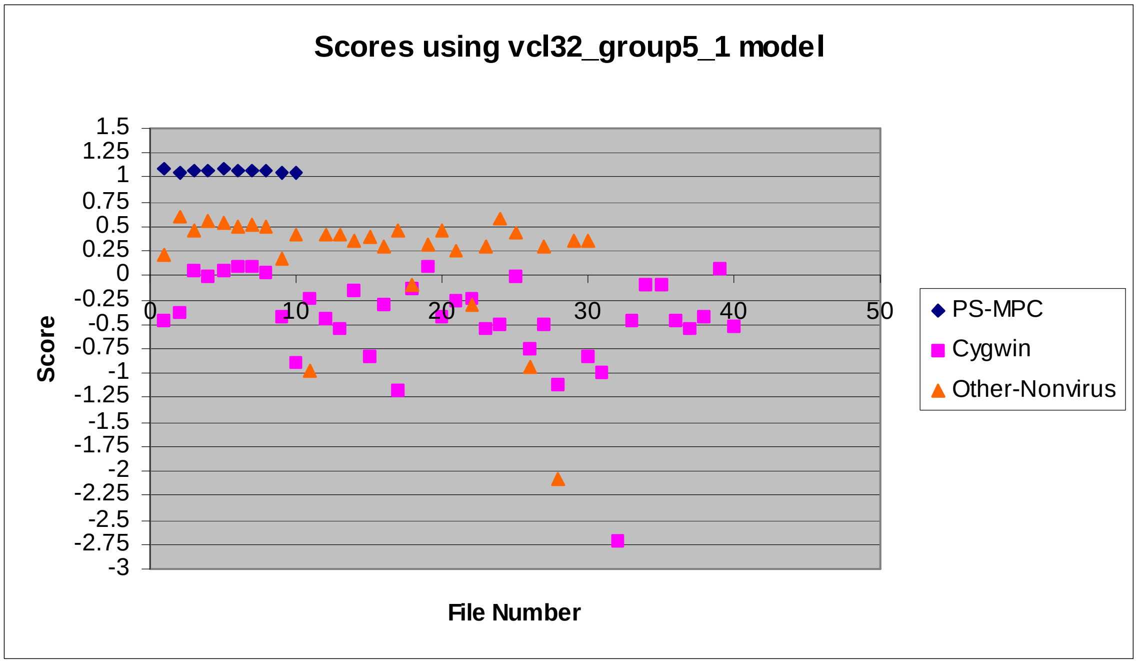
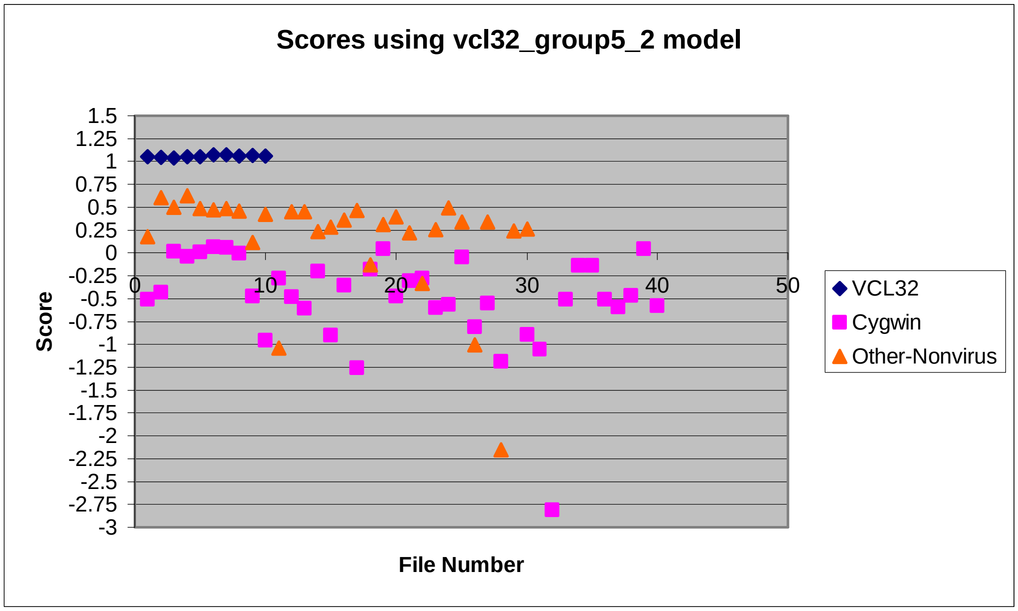


Table A-2 Scores of Virus and Non Virus files using vcl32_group5_2 model

VCL32 Virus Variants		Non Virus Files			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
vcl32_01	1.054748	cygwin_01	-0.510939	nonvirus_01	0.175959
vcl32_02	1.041679	cygwin_02	-0.429031	nonvirus_02	0.607093
vcl32_03	1.038289	cygwin_03	0.018187	nonvirus_03	0.500238
vcl32_04	1.050418	cygwin_04	-0.041686	nonvirus_04	0.62645
vcl32_05	1.051996	cygwin_05	0.00586	nonvirus_05	0.482649
vcl32_06	1.076125	cygwin_06	0.068762	nonvirus_06	0.469946
vcl32_07	1.071717	cygwin_07	0.05598	nonvirus_07	0.481795
vcl32_08	1.057444	cygwin_08	-0.001187	nonvirus_08	0.459852
vcl32_09	1.067382	cygwin_09	-0.470955	nonvirus_09	0.115241
vcl32_10	1.056705	cygwin_10	-0.954708	nonvirus_10	0.423541
		cygwin_11	-0.280892	nonvirus_11	-1.041574
		cygwin_12	-0.483825	nonvirus_12	0.447212
		cygwin_13	-0.603847	nonvirus_13	0.447212
		cygwin_14	-0.201867	nonvirus_14	0.236376
		cygwin_15	-0.89825	nonvirus_15	0.284199
		cygwin_16	-0.356652	nonvirus_16	0.359028
		cygwin_17	-1.259348	nonvirus_17	0.464545
		cygwin_18	-0.178455	nonvirus_18	-0.12838
		cygwin_19	0.043298	nonvirus_19	0.308425
		cygwin_20	-0.473163	nonvirus_20	0.394181
		cygwin_21	-0.307048	nonvirus_21	0.222292
		cygwin_22	-0.280265	nonvirus_22	-0.334553
		cygwin_23	-0.600964	nonvirus_23	0.257425
		cygwin_24	-0.56236	nonvirus_24	0.494217
		cygwin_25	-0.049662	nonvirus_25	0.338486
		cygwin_26	-0.810152	nonvirus_26	-1.005699
		cygwin_27	-0.550994	nonvirus_27	0.340329
		cygwin_28	-1.187329	nonvirus_28	-2.154028
		cygwin_29	-6.570453	nonvirus_29	0.240242
		cygwin_30	-0.892495	nonvirus_30	0.261265
		cygwin_31	-1.053905		
		cygwin_32	-2.814226		
		cygwin_33	-0.511613		
		cygwin_34	-0.136853		
		cygwin_35	-0.13808		
		cygwin_36	-0.506485		
		cygwin_37	-0.593724		
		cygwin_38	-0.464666		
		cygwin_39	0.040891		
		cygwin_40	-0.579538		

Figure A-2: Graphical representation of Virus and Non-Virus Scores using vcl32_group5_2 model



APPENDIX B - PS-MPC Scores

Table B-1 Scores of Virus and Non Virus files using psmpc_group10_1 model

PSPMC Virus Variants		Non Virus Files			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
psmpc_01	1.323747	cygwin_01	0.217836	nonvirus_01	-0.17126
psmpc_02	1.621965	cygwin_02	0.278389	nonvirus_02	-0.035853
psmpc_03	1.54293	cygwin_03	0.137888	nonvirus_03	-0.094112
psmpc_04	1.02367	cygwin_04	0.203186	nonvirus_04	0.187106
psmpc_05	1.587549	cygwin_05	0.113871	nonvirus_05	-0.168395
psmpc_06	1.524759	cygwin_06	0.106767	nonvirus_06	-0.113968
psmpc_07	0.922988	cygwin_07	0.099252	nonvirus_07	-0.130918
psmpc_08	1.621965	cygwin_08	0.122255	nonvirus_08	-0.119984
psmpc_09	1.385606	cygwin_09	0.107664	nonvirus_09	-0.05732
psmpc_10	0.961724	cygwin_10	0.304064	nonvirus_10	-0.118333
psmpc_11	0.873914	cygwin_11	0.207124	nonvirus_11	-0.056218
psmpc_12	0.943829	cygwin_12	0.175749	nonvirus_12	-0.088344
psmpc_13	0.962353	cygwin_13	0.118547	nonvirus_13	-0.141422
psmpc_14	1.403483	cygwin_14	0.109732	nonvirus_14	-0.218387
psmpc_15	1.379162	cygwin_15	0.263593	nonvirus_15	-0.203497
psmpc_16	1.45283	cygwin_16	0.289688	nonvirus_16	0.015157
psmpc_17	1.009983	cygwin_17	0.194993	nonvirus_17	-0.100559
psmpc_18	1.605451	cygwin_18	0.247258	nonvirus_18	-0.102171
psmpc_19	1.40997	cygwin_19	0.167704	nonvirus_19	-0.130722
psmpc_20	1.621965	cygwin_20	0.138071	nonvirus_20	-0.218612
psmpc_21	1.607687	cygwin_21	0.234471	nonvirus_21	-0.1514
psmpc_22	0.958344	cygwin_22	0.267159	nonvirus_22	-0.050515
psmpc_23	1.614169	cygwin_23	0.01101	nonvirus_23	-0.286356
psmpc_24	1.610268	cygwin_24	0.204981	nonvirus_24	-0.19157
psmpc_25	1.030705	cygwin_25	0.158373	nonvirus_25	-0.235362
psmpc_26	1.017315	cygwin_26	0.171962	nonvirus_26	0.233872
psmpc_27	1.340959	cygwin_27	0.192007	nonvirus_27	0.051087
psmpc_28	1.520831	cygwin_28	0.261288	nonvirus_28	0.041697
psmpc_29	0.949162	cygwin_29	0.311014	nonvirus_29	-0.220485
psmpc_30	1.589719	cygwin_30	0.191735	nonvirus_30	-0.210733
		cygwin_31	0.310988		
		cygwin_32	0.23574		
		cygwin_33	0.151786		
		cygwin_34	0.221324		
		cygwin_35	0.135578		
		cygwin_36	0.222211		
		cygwin_37	0.223585		
		cygwin_38	0.164705		
		cygwin_39	0.24573		
		cygwin_40	0.275728		

Figure B-1: Graphical representation of Virus and Non-Virus Scores using psmpc_group10_1 model

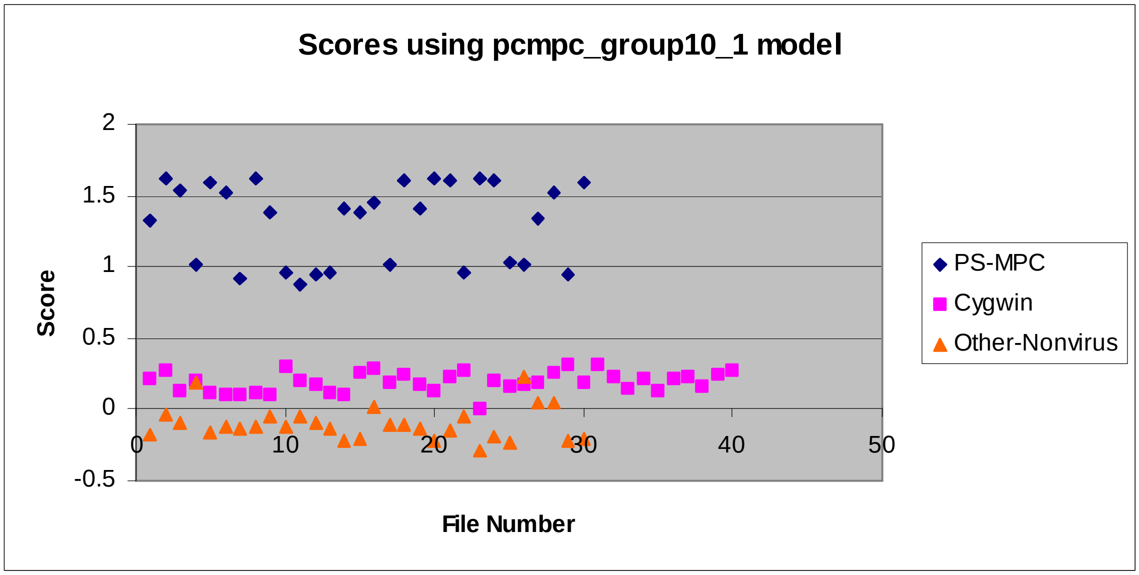


Table B-2 Scores of Virus and Non Virus files using psmpc_group10_2 model

PSPMC Virus Variants		Non Virus Files			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
psmpc_01	1.299699	cygwin_01	0.60396	nonvirus_01	-0.011451
psmpc_02	1.499945	cygwin_02	0.743039	nonvirus_02	0.222913
psmpc_03	1.426114	cygwin_03	0.453493	nonvirus_03	0.19182
psmpc_04	1.012006	cygwin_04	0.569435	nonvirus_04	0.581364
psmpc_05	1.464344	cygwin_05	0.407526	nonvirus_05	0.010183
psmpc_06	1.447789	cygwin_06	0.37784	nonvirus_06	0.106866
psmpc_07	0.875303	cygwin_07	0.389348	nonvirus_07	0.053376
psmpc_08	1.499945	cygwin_08	0.414411	nonvirus_08	0.088721
psmpc_09	1.364755	cygwin_09	0.379526	nonvirus_09	0.115241
psmpc_10	1.07822	cygwin_10	0.667799	nonvirus_10	0.122428
psmpc_11	1.006404	cygwin_11	0.569719	nonvirus_11	0.248631
psmpc_12	1.093912	cygwin_12	0.489344	nonvirus_12	0.186405
psmpc_13	1.074151	cygwin_13	0.369263	nonvirus_13	0.196976
psmpc_14	1.383742	cygwin_14	0.379498	nonvirus_14	0.039328
psmpc_15	1.367184	cygwin_15	0.665051	nonvirus_15	0.049431
psmpc_16	1.373806	cygwin_16	0.688913	nonvirus_16	0.285888
psmpc_17	1.023055	cygwin_17	0.508443	nonvirus_17	0.104013
psmpc_18	1.495992	cygwin_18	0.644309	nonvirus_18	0.134148
psmpc_19	1.381297	cygwin_19	0.507705	nonvirus_19	0.057854
psmpc_20	1.499945	cygwin_20	0.390515	nonvirus_20	-0.001147
psmpc_21	1.489273	cygwin_21	0.546206	nonvirus_21	-0.036187
psmpc_22	0.927391	cygwin_22	0.660234	nonvirus_22	0.168131
psmpc_23	1.492649	cygwin_23	0.205626	nonvirus_23	-0.181052
psmpc_24	1.494176	cygwin_24	0.533397	nonvirus_24	-0.029677
psmpc_25	1.033471	cygwin_25	0.506076	nonvirus_25	-0.083614
psmpc_26	1.023167	cygwin_26	0.438186	nonvirus_26	0.489644
psmpc_27	1.325211	cygwin_27	0.468334	nonvirus_27	0.43179
psmpc_28	1.43689	cygwin_28	0.556737	nonvirus_28	0.287961
psmpc_29	1.077357	cygwin_29	0.377149	nonvirus_29	0.067728
psmpc_30	1.476733	cygwin_30	0.397222	nonvirus_30	-0.090047
		cygwin_31	0.687407		
		cygwin_32	0.448065		
		cygwin_33	0.447832		
		cygwin_34	0.649103		
		cygwin_35	0.458314		
		cygwin_36	0.616491		
		cygwin_37	0.564271		
		cygwin_38	0.508992		
		cygwin_39	0.673767		
		cygwin_40	0.64151		

Figure B-2: Graphical representation of Virus and Non-Virus Scores using psmpc_group10_2 model

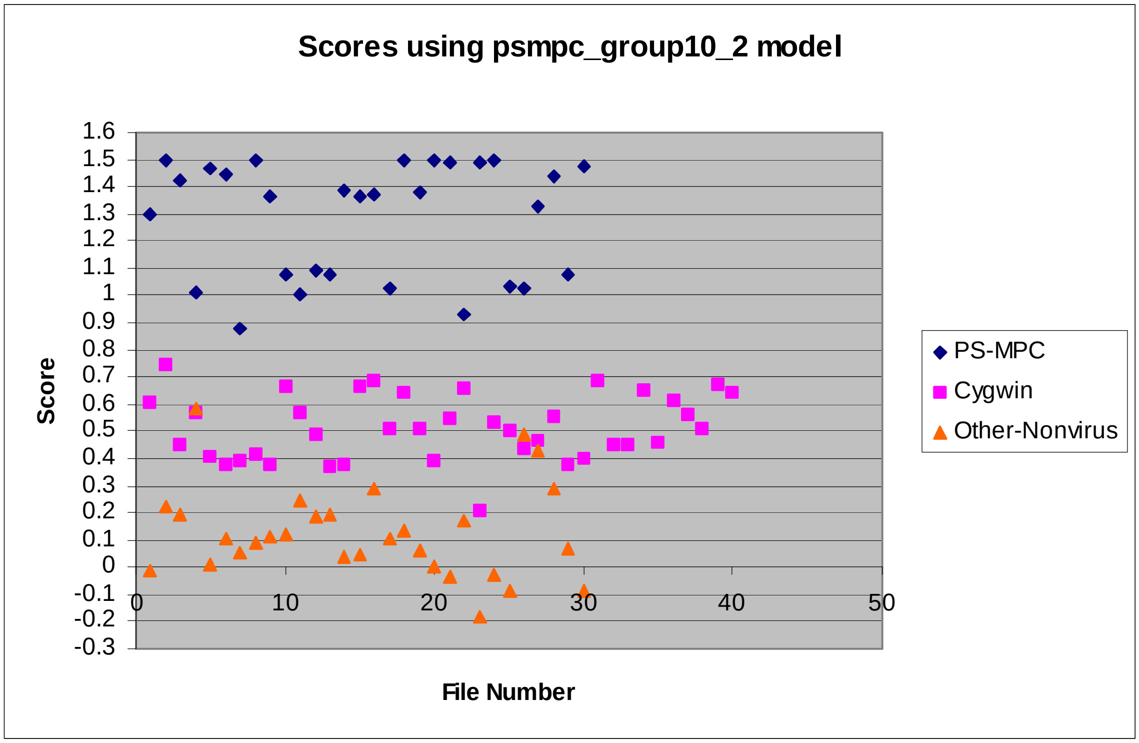
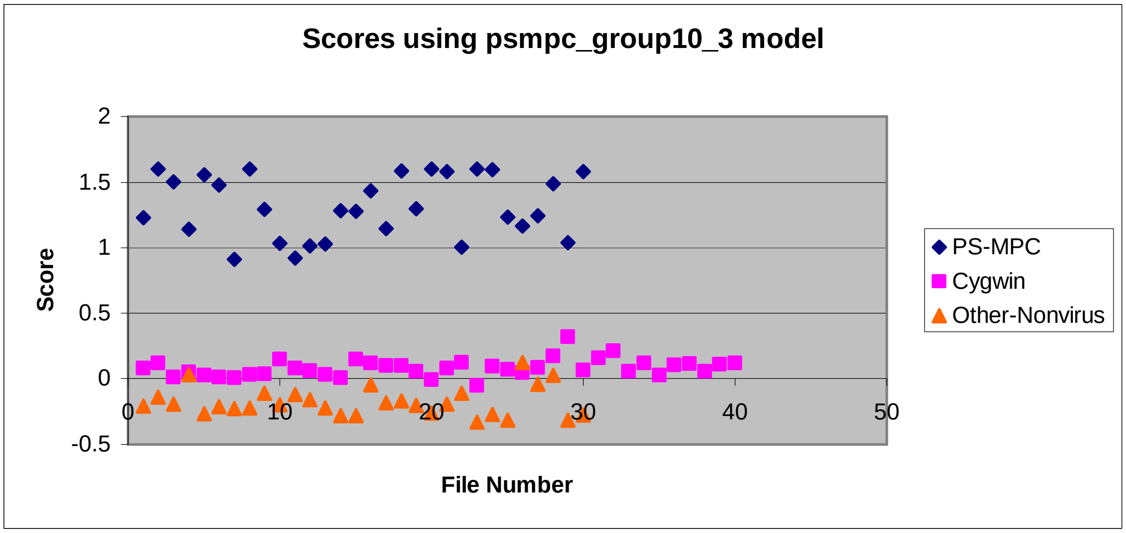


Table B-3 Scores of Virus and Non Virus files using psmpc_group10_3 model

PSPMC Virus Variants		Non Virus Files			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
psmpc_01	1.227648	cygwin_01	0.08141	nonvirus_01	-0.208046
psmpc_02	1.600759	cygwin_02	0.12026	nonvirus_02	-0.140612
psmpc_03	1.505053	cygwin_03	0.013068	nonvirus_03	-0.191327
psmpc_04	1.144719	cygwin_04	0.05019	nonvirus_04	0.031832
psmpc_05	1.554167	cygwin_05	0.024026	nonvirus_05	-0.266684
psmpc_06	1.476359	cygwin_06	0.013583	nonvirus_06	-0.211736
psmpc_07	0.910976	cygwin_07	0.004608	nonvirus_07	-0.224777
psmpc_08	1.600759	cygwin_08	0.032545	nonvirus_08	-0.221471
psmpc_09	1.294007	cygwin_09	0.035636	nonvirus_09	-0.111186
psmpc_10	1.035318	cygwin_10	0.149571	nonvirus_10	-0.199914
psmpc_11	0.923018	cygwin_11	0.080839	nonvirus_11	-0.118773
psmpc_12	1.015707	cygwin_12	0.058602	nonvirus_12	-0.159612
psmpc_13	1.028569	cygwin_13	0.032574	nonvirus_13	-0.220773
psmpc_14	1.282105	cygwin_14	0.006599	nonvirus_14	-0.279282
psmpc_15	1.278729	cygwin_15	0.148496	nonvirus_15	-0.279647
psmpc_16	1.435184	cygwin_16	0.121077	nonvirus_16	-0.046356
psmpc_17	1.147134	cygwin_17	0.098734	nonvirus_17	-0.183069
psmpc_18	1.58493	cygwin_18	0.101414	nonvirus_18	-0.167831
psmpc_19	1.297483	cygwin_19	0.057787	nonvirus_19	-0.201428
psmpc_20	1.600759	cygwin_20	-0.005496	nonvirus_20	-0.260167
psmpc_21	1.582813	cygwin_21	0.081491	nonvirus_21	-0.193507
psmpc_22	1.006626	cygwin_22	0.122084	nonvirus_22	-0.109163
psmpc_23	1.600967	cygwin_23	-0.052099	nonvirus_23	-0.331173
psmpc_24	1.596352	cygwin_24	0.096249	nonvirus_24	-0.271952
psmpc_25	1.232587	cygwin_25	0.07178	nonvirus_25	-0.313998
psmpc_26	1.164333	cygwin_26	0.04778	nonvirus_26	0.122739
psmpc_27	1.242206	cygwin_27	0.08651	nonvirus_27	-0.042634
psmpc_28	1.489333	cygwin_28	0.174561	nonvirus_28	0.024008
psmpc_29	1.042142	cygwin_29	0.316948	nonvirus_29	-0.31655
psmpc_30	1.57858	cygwin_30	0.066092	nonvirus_30	-0.275571
		cygwin_31	0.155816		
		cygwin_32	0.21262		
		cygwin_33	0.054895		
		cygwin_34	0.120884		
		cygwin_35	0.028028		
		cygwin_36	0.106603		
		cygwin_37	0.112139		
		cygwin_38	0.055971		
		cygwin_39	0.110219		
		cygwin_40	0.117183		

Figure B-3: Graphical representation of Virus and Non-Virus Scores using psmpc_group10_3 model



APPENDIX C - NGVCK Scores

Table C-1.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_01 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.860894	cygwin_01	0.610366	nonvirus_01	0.293974
ngvck_002	0.868975	cygwin_02	**0.755483**	nonvirus_02	0.510075
ngvck_003	1.000545	cygwin_03	0.547281	nonvirus_03	0.408427
ngvck_004	0.870732	cygwin_04	0.594871	nonvirus_04	**0.747985**
ngvck_005	0.810336	cygwin_05	0.531224	nonvirus_05	0.332839
ngvck_006	0.867058	cygwin_06	0.521635	nonvirus_06	0.415972
ngvck_007	0.846234	cygwin_07	0.520335	nonvirus_07	0.359667
ngvck_008	0.794665	cygwin_08	0.581491	nonvirus_08	0.402952
ngvck_009	0.9029	cygwin_09	0.505439	nonvirus_09	0.246162
ngvck_010	0.964697	cygwin_10	0.627607	nonvirus_10	0.419246
ngvck_011	0.820068	cygwin_11	0.520347	nonvirus_11	0.4466
ngvck_012	0.946846	cygwin_12	0.519592	nonvirus_12	0.470656
ngvck_013	0.890484	cygwin_13	0.462797	nonvirus_13	0.517136
ngvck_014	0.819489	cygwin_14	0.416628	nonvirus_14	0.313303
ngvck_015	0.904151	cygwin_15	0.622661	nonvirus_15	0.334759
ngvck_016	0.946656	cygwin_16	**0.685346**	nonvirus_16	0.43101
ngvck_017	0.822826	cygwin_17	0.511617	nonvirus_17	0.389576
ngvck_018	0.793125	cygwin_18	0.65049	nonvirus_18	0.502135
ngvck_019	0.86738	cygwin_19	0.525175	nonvirus_19	0.406563
ngvck_020	**0.573609**	cygwin_20	0.446541	nonvirus_20	0.274402
ngvck_021	0.841805	cygwin_21	0.558141	nonvirus_21	0.257406
ngvck_022	0.789624	cygwin_22	0.617675	nonvirus_22	0.436877
ngvck_023	0.805843	cygwin_23	0.471552	nonvirus_23	0.106282
ngvck_024	0.772065	cygwin_24	0.493804	nonvirus_24	0.327661
ngvck_025	0.77012	cygwin_25	0.479633	nonvirus_25	0.195072
ngvck_026	0.821852	cygwin_26	0.506057	nonvirus_26	-2.58214
ngvck_027	0.84134	cygwin_27	0.507927	nonvirus_27	0.566432
ngvck_028	0.807432	cygwin_28	0.591615	nonvirus_28	0.327207
ngvck_029	0.799459	cygwin_29	0.166759	nonvirus_29	0.346006
ngvck_030	0.755152	cygwin_30	0.463929	nonvirus_30	0.249738
ngvck_031	0.85008	cygwin_31	**0.686945**		
ngvck_032	0.757738	cygwin_32	0.460027		
ngvck_033	0.859768	cygwin_33	0.528198		
ngvck_034	0.792964	cygwin_34	**0.675175**		
ngvck_035	0.723463	cygwin_35	0.536658		
ngvck_036	0.81013	cygwin_36	0.628225		
ngvck_037	0.846603	cygwin_37	0.563168		
ngvck_038	0.727694	cygwin_38	0.599896		
ngvck_039	0.840671	cygwin_39	**0.67509**		
ngvck_040	0.843615	cygwin_40	0.595222		

Table C-1.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_01 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.753635	ngvck_081	0.862261	ngvck_121	0.804187	ngvck_161	0.857069
ngvck_042	0.871583	ngvck_082	0.804648	ngvck_122	0.8091	ngvck_162	0.817811
ngvck_043	0.842921	ngvck_083	0.780752	ngvck_123	0.814295	ngvck_163	0.834065
ngvck_044	0.817743	ngvck_084	0.840109	ngvck_124	0.825707	ngvck_164	0.858248
ngvck_045	0.797452	ngvck_085	0.713844	ngvck_125	0.709777	ngvck_165	0.745027
ngvck_046	0.833126	ngvck_086	0.892858	ngvck_126	0.898773	ngvck_166	0.875788
ngvck_047	0.921651	ngvck_087	0.790193	ngvck_127	0.739335	ngvck_167	0.813439
ngvck_048	0.869399	ngvck_088	0.815063	ngvck_128	0.72748	ngvck_168	0.79234
ngvck_049	0.882094	ngvck_089	0.763562	ngvck_129	0.706566	ngvck_169	0.8794
ngvck_050	0.756481	ngvck_090	0.792515	ngvck_130	0.864073	ngvck_170	0.768415
ngvck_051	0.761084	ngvck_091	0.82744	ngvck_131	0.805538	ngvck_171	0.796446
ngvck_052	0.825251	ngvck_092	0.748596	ngvck_132	0.916603	ngvck_172	0.852494
ngvck_053	0.892163	ngvck_093	0.752862	ngvck_133	0.85658	ngvck_173	0.799862
ngvck_054	0.856581	ngvck_094	0.756383	ngvck_134	0.866253	ngvck_174	0.660757
ngvck_055	0.907063	ngvck_095	0.776757	ngvck_135	0.752514	ngvck_175	0.78357
ngvck_056	0.864343	ngvck_096	0.834515	ngvck_136	0.861997	ngvck_176	0.890955
ngvck_057	**0.655816**	ngvck_097	0.880577	ngvck_137	0.857547	ngvck_177	0.839373
ngvck_058	0.821135	ngvck_098	0.839838	ngvck_138	0.816845	ngvck_178	0.750812
ngvck_059	0.884584	ngvck_099	0.78702	ngvck_139	0.766686	ngvck_179	0.800548
ngvck_060	0.907645	ngvck_100	0.783416	ngvck_140	0.838792	ngvck_180	0.925601
ngvck_061	0.833157	ngvck_101	0.843918	ngvck_141	0.724386	ngvck_181	0.797583
ngvck_062	0.831591	ngvck_102	0.770927	ngvck_142	0.838825	ngvck_182	0.829643
ngvck_063	0.84798	ngvck_103	0.840213	ngvck_143	0.762811	ngvck_183	0.838471
ngvck_064	0.820833	ngvck_104	0.849388	ngvck_144	0.770057	ngvck_184	0.813208
ngvck_065	0.87009	ngvck_105	0.829788	ngvck_145	0.807744	ngvck_185	0.895381
ngvck_066	0.751655	ngvck_106	0.746036	ngvck_146	0.821296	ngvck_186	0.827445
ngvck_067	0.805768	ngvck_107	0.751918	ngvck_147	0.842212	ngvck_187	0.777924
ngvck_068	0.881451	ngvck_108	0.848619	ngvck_148	0.872007	ngvck_188	0.870205
ngvck_069	0.812108	ngvck_109	0.878091	ngvck_149	0.803412	ngvck_189	0.852584
ngvck_070	0.780337	ngvck_110	0.907019	ngvck_150	0.749523	ngvck_190	0.814617
ngvck_071	0.786372	ngvck_111	0.801105	ngvck_151	0.778977	ngvck_191	0.773475
ngvck_072	0.764434	ngvck_112	0.831256	ngvck_152	0.867348	ngvck_192	0.81144
ngvck_073	0.681482	ngvck_113	0.759036	ngvck_153	0.8419	ngvck_193	0.854805
ngvck_074	0.85031	ngvck_114	0.817647	ngvck_154	0.844562	ngvck_194	0.848243
ngvck_075	0.794	ngvck_115	0.783875	ngvck_155	0.913228	ngvck_195	0.864787
ngvck_076	0.78672	ngvck_116	0.761749	ngvck_156	0.75372	ngvck_196	0.762374
ngvck_077	0.829067	ngvck_117	0.860347	ngvck_157	0.873416	ngvck_197	0.813457
ngvck_078	0.904401	ngvck_118	0.797725	ngvck_158	0.775889	ngvck_198	0.74458
ngvck_079	0.853988	ngvck_119	0.885495	ngvck_159	0.816867	ngvck_199	0.84178
ngvck_080	0.792293	ngvck_120	0.759682	ngvck_160	0.848925	ngvck_200	1.030041

Figure C-1: Graphical representation of Virus and Non-Virus Scores using ngvck_ pp_group20_01 model

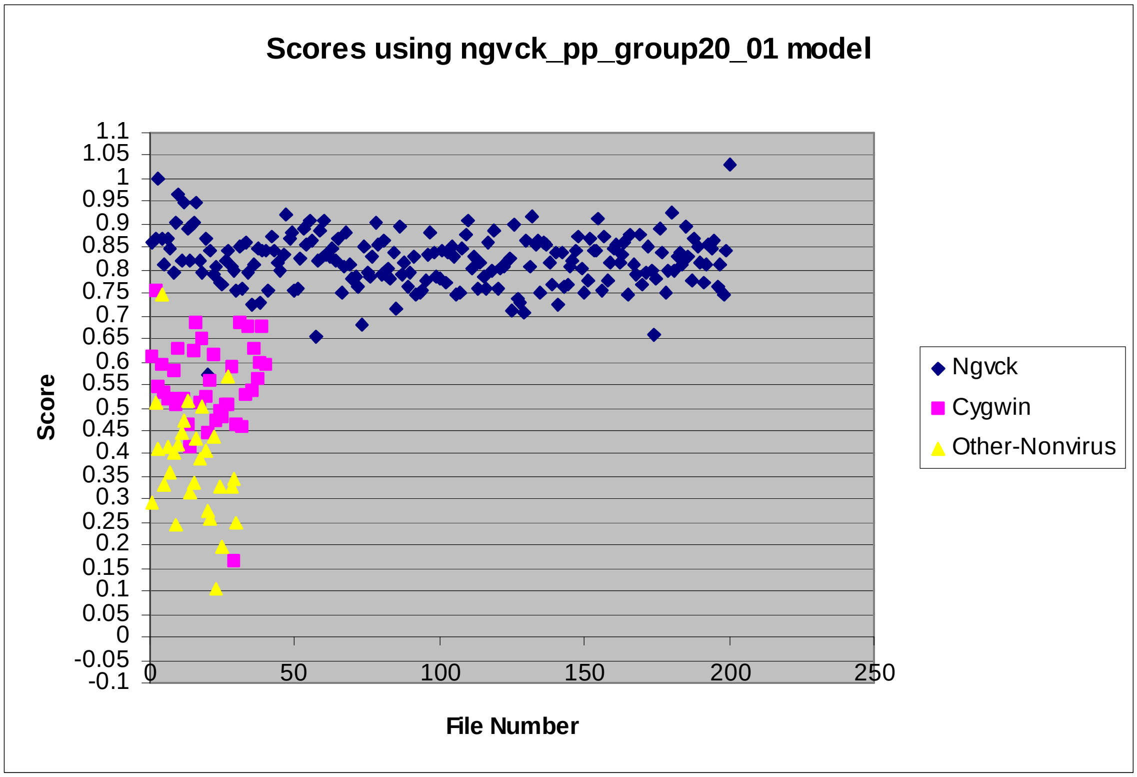


Table C-2.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_02 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.828381	cygwin_01	0.580517	nonvirus_01	0.169417
ngvck_002	0.757786	cygwin_02	**0.76851**	nonvirus_02	0.558092
ngvck_003	0.847624	cygwin_03	0.511929	nonvirus_03	0.448329
ngvck_004	0.7368	cygwin_04	0.613065	nonvirus_04	**0.724204**
ngvck_005	0.743113	cygwin_05	0.530385	nonvirus_05	0.35404
ngvck_006	0.752671	cygwin_06	0.515502	nonvirus_06	0.428108
ngvck_007	0.798293	cygwin_07	0.518819	nonvirus_07	0.36013
ngvck_008	0.736816	cygwin_08	0.540779	nonvirus_08	0.411188
ngvck_009	0.796243	cygwin_09	0.453575	nonvirus_09	0.233132
ngvck_010	0.800729	cygwin_10	0.656175	nonvirus_10	0.413499
ngvck_011	0.747784	cygwin_11	0.492416	nonvirus_11	0.466024
ngvck_012	0.771351	cygwin_12	0.561099	nonvirus_12	0.392701
ngvck_013	0.801882	cygwin_13	0.506445	nonvirus_13	0.523534
ngvck_014	0.674445	cygwin_14	0.408331	nonvirus_14	0.351599
ngvck_015	0.771006	cygwin_15	0.619902	nonvirus_15	0.3197
ngvck_016	0.822784	cygwin_16	**0.712102**	nonvirus_16	0.288495
ngvck_017	0.761653	cygwin_17	0.543472	nonvirus_17	0.408147
ngvck_018	0.703043	cygwin_18	**0.683028**	nonvirus_18	0.4351
ngvck_019	0.802737	cygwin_19	0.488043	nonvirus_19	0.313088
ngvck_020	**0.624427**	cygwin_20	0.487227	nonvirus_20	0.2605
ngvck_021	0.866153	cygwin_21	0.582314	nonvirus_21	0.235216
ngvck_022	0.730291	cygwin_22	0.526208	nonvirus_22	0.452235
ngvck_023	0.870428	cygwin_23	0.461065	nonvirus_23	0.039321
ngvck_024	0.812	cygwin_24	0.463993	nonvirus_24	0.319042
ngvck_025	0.75809	cygwin_25	0.511801	nonvirus_25	0.308221
ngvck_026	0.85087	cygwin_26	0.499626	nonvirus_26	-2.662959
ngvck_027	0.834567	cygwin_27	0.523655	nonvirus_27	0.515032
ngvck_028	0.917285	cygwin_28	0.572406	nonvirus_28	0.327243
ngvck_029	0.880627	cygwin_29	0.144156	nonvirus_29	0.395129
ngvck_030	0.786614	cygwin_30	0.488398	nonvirus_30	0.274813
ngvck_031	0.830041	cygwin_31	**0.703662**		
ngvck_032	0.77468	cygwin_32	0.487267		
ngvck_033	0.884885	cygwin_33	0.494805		
ngvck_034	0.807775	cygwin_34	0.598537		
ngvck_035	0.765953	cygwin_35	0.479335		
ngvck_036	0.818434	cygwin_36	0.534515		
ngvck_037	0.781235	cygwin_37	0.555427		
ngvck_038	0.858023	cygwin_38	0.51689		
ngvck_039	0.854824	cygwin_39	0.531798		
ngvck_040	0.771913	cygwin_40	0.609794		

Table C-2.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_02 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.738522	ngvck_081	0.824589	ngvck_121	0.738525	ngvck_161	0.799477
ngvck_042	0.845601	ngvck_082	0.770465	ngvck_122	0.771785	ngvck_162	0.786537
ngvck_043	0.796762	ngvck_083	0.757968	ngvck_123	0.73068	ngvck_163	0.754703
ngvck_044	0.704713	ngvck_084	0.81003	ngvck_124	0.763845	ngvck_164	0.794176
ngvck_045	0.815328	ngvck_085	0.686612	ngvck_125	0.695978	ngvck_165	0.672203
ngvck_046	0.790095	ngvck_086	0.774511	ngvck_126	0.833132	ngvck_166	0.831769
ngvck_047	0.838143	ngvck_087	0.740871	ngvck_127	0.697748	ngvck_167	0.783127
ngvck_048	0.761381	ngvck_088	0.713647	ngvck_128	0.723479	ngvck_168	0.703943
ngvck_049	0.815258	ngvck_089	0.737004	ngvck_129	0.685144	ngvck_169	0.763457
ngvck_050	0.74238	ngvck_090	0.783163	ngvck_130	0.768987	ngvck_170	0.760824
ngvck_051	0.675937	ngvck_091	0.822124	ngvck_131	0.806899	ngvck_171	0.769112
ngvck_052	0.72168	ngvck_092	0.738471	ngvck_132	0.833974	ngvck_172	0.80619
ngvck_053	0.863495	ngvck_093	0.775828	ngvck_133	0.742502	ngvck_173	0.698021
ngvck_054	0.756973	ngvck_094	0.739925	ngvck_134	0.794522	ngvck_174	**0.641684**
ngvck_055	0.802474	ngvck_095	0.727793	ngvck_135	0.696242	ngvck_175	0.720702
ngvck_056	0.790627	ngvck_096	0.740935	ngvck_136	0.763223	ngvck_176	0.819771
ngvck_057	0.672291	ngvck_097	0.834719	ngvck_137	0.827659	ngvck_177	0.787317
ngvck_058	0.780045	ngvck_098	0.80435	ngvck_138	0.787525	ngvck_178	0.670883
ngvck_059	0.821721	ngvck_099	0.760884	ngvck_139	0.732842	ngvck_179	0.766033
ngvck_060	0.857707	ngvck_100	0.760663	ngvck_140	0.779013	ngvck_180	0.825609
ngvck_061	0.800807	ngvck_101	0.716262	ngvck_141	0.739274	ngvck_181	0.804958
ngvck_062	0.808945	ngvck_102	0.752558	ngvck_142	0.736926	ngvck_182	0.772727
ngvck_063	0.786805	ngvck_103	0.752348	ngvck_143	0.754939	ngvck_183	0.798342
ngvck_064	0.801636	ngvck_104	0.803709	ngvck_144	0.729938	ngvck_184	0.763681
ngvck_065	0.766359	ngvck_105	0.765567	ngvck_145	0.759534	ngvck_185	0.845407
ngvck_066	0.660153	ngvck_106	0.730209	ngvck_146	0.78559	ngvck_186	0.79659
ngvck_067	0.7411	ngvck_107	0.732221	ngvck_147	0.803442	ngvck_187	0.730319
ngvck_068	0.827916	ngvck_108	0.806108	ngvck_148	0.804933	ngvck_188	0.760496
ngvck_069	0.762093	ngvck_109	0.805707	ngvck_149	0.784507	ngvck_189	0.755723
ngvck_070	0.756316	ngvck_110	0.804795	ngvck_150	0.731564	ngvck_190	0.684666
ngvck_071	0.758477	ngvck_111	0.782849	ngvck_151	0.737131	ngvck_191	0.76374
ngvck_072	0.730391	ngvck_112	0.759181	ngvck_152	0.791855	ngvck_192	0.75298
ngvck_073	0.667772	ngvck_113	0.73387	ngvck_153	0.775319	ngvck_193	0.794093
ngvck_074	0.818807	ngvck_114	0.803494	ngvck_154	0.771986	ngvck_194	0.814975
ngvck_075	0.774266	ngvck_115	0.762706	ngvck_155	0.826642	ngvck_195	0.750904
ngvck_076	0.746308	ngvck_116	0.729938	ngvck_156	0.725358	ngvck_196	0.727907
ngvck_077	0.791255	ngvck_117	0.74896	ngvck_157	0.825593	ngvck_197	0.767195
ngvck_078	0.834478	ngvck_118	0.732935	ngvck_158	0.696075	ngvck_198	0.764788
ngvck_079	0.791101	ngvck_119	0.781585	ngvck_159	0.809949	ngvck_199	0.740886
ngvck_080	0.764665	ngvck_120	0.767033	ngvck_160	0.771337	ngvck_200	0.785756

Figure C-2: Graphical representation of Virus and Non-Virus Scores using ngvck_ pp_group20_02 model

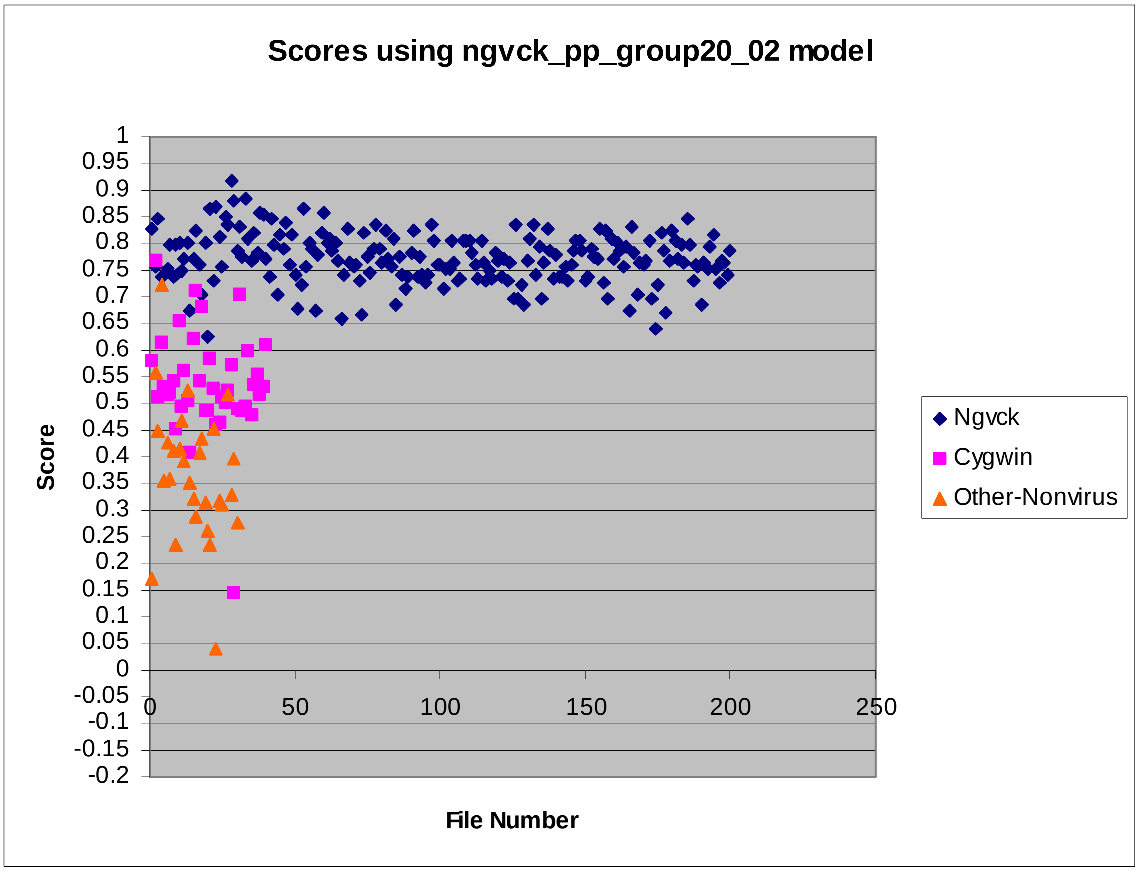


Table C-3.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_03 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.841463	cygwin_01	**0.734867**	nonvirus_01	0.364686
ngvck_002	0.890275	cygwin_02	**0.900736**	nonvirus_02	**0.823921**
ngvck_003	0.908429	cygwin_03	**0.750114**	nonvirus_03	**0.772905**
ngvck_004	0.931931	cygwin_04	**0.814867**	nonvirus_04	**1.016538**
ngvck_005	0.816886	cygwin_05	**0.748111**	nonvirus_05	0.659697
ngvck_006	0.836098	cygwin_06	**0.757009**	nonvirus_06	**0.717304**
ngvck_007	0.829136	cygwin_07	**0.753734**	nonvirus_07	0.672208
ngvck_008	0.805622	cygwin_08	**0.753501**	nonvirus_08	**0.703686**
ngvck_009	0.873528	cygwin_09	0.562229	nonvirus_09	0.576922
ngvck_010	0.932081	cygwin_10	**0.777756**	nonvirus_10	**0.703593**
ngvck_011	0.851462	cygwin_11	0.684623	nonvirus_11	0.615317
ngvck_012	0.892772	cygwin_12	**0.732611**	nonvirus_12	0.687278
ngvck_013	0.819372	cygwin_13	0.638832	nonvirus_13	**0.810699**
ngvck_014	0.805064	cygwin_14	0.587021	nonvirus_14	0.619772
ngvck_015	0.93064	cygwin_15	**0.737692**	nonvirus_15	0.619115
ngvck_016	0.871456	cygwin_16	**0.863374**	nonvirus_16	0.583951
ngvck_017	0.787787	cygwin_17	0.619695	nonvirus_17	0.668209
ngvck_018	0.788396	cygwin_18	**0.841042**	nonvirus_18	0.637316
ngvck_019	0.86655	cygwin_19	**0.765583**	nonvirus_19	0.574846
ngvck_020	**0.573397**	cygwin_20	0.609147	nonvirus_20	0.579644
ngvck_021	0.849945	cygwin_21	**0.760312**	nonvirus_21	0.490828
ngvck_022	0.892437	cygwin_22	0.679325	nonvirus_22	0.630619
ngvck_023	0.841527	cygwin_23	0.656278	nonvirus_23	0.346088
ngvck_024	0.797918	cygwin_24	0.630473	nonvirus_24	0.64141
ngvck_025	0.738444	cygwin_25	0.473591	nonvirus_25	0.511113
ngvck_026	0.824084	cygwin_26	0.595079	nonvirus_26	-2.575797
ngvck_027	0.845827	cygwin_27	0.647573	nonvirus_27	**0.763262**
ngvck_028	0.834182	cygwin_28	0.692067	nonvirus_28	0.375109
ngvck_029	0.813924	cygwin_29	0.139012	nonvirus_29	0.612437
ngvck_030	0.783003	cygwin_30	0.579673	nonvirus_30	0.540981
ngvck_031	0.888515	cygwin_31	**0.81042**		
ngvck_032	0.779469	cygwin_32	0.500726		
ngvck_033	0.84003	cygwin_33	0.67757		
ngvck_034	0.864777	cygwin_34	**0.747324**		
ngvck_035	0.745495	cygwin_35	0.636384		
ngvck_036	0.916553	cygwin_36	0.665231		
ngvck_037	0.924546	cygwin_37	0.668131		
ngvck_038	0.728479	cygwin_38	0.648995		
ngvck_039	0.872537	cygwin_39	**0.785913**		
ngvck_040	1.031087	cygwin_40	**0.744111**		

Table C-3.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_03 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.83925	ngvck_081	0.872857	ngvck_121	0.797442	ngvck_161	0.924546
ngvck_042	0.91401	ngvck_082	0.808906	ngvck_122	0.805456	ngvck_162	0.849196
ngvck_043	0.8625	ngvck_083	0.783196	ngvck_123	0.851486	ngvck_163	0.928015
ngvck_044	0.922903	ngvck_084	0.86801	ngvck_124	0.895327	ngvck_164	0.934493
ngvck_045	0.969751	ngvck_085	0.71809	ngvck_125	0.754162	ngvck_165	0.786579
ngvck_046	0.845309	ngvck_086	0.881484	ngvck_126	0.85057	ngvck_166	0.884516
ngvck_047	1.087423	ngvck_087	0.846087	ngvck_127	0.754502	ngvck_167	0.820076
ngvck_048	1.014962	ngvck_088	0.882405	ngvck_128	0.734585	ngvck_168	0.863464
ngvck_049	0.912537	ngvck_089	0.805833	ngvck_129	0.798054	ngvck_169	0.918203
ngvck_050	0.818697	ngvck_090	0.904509	ngvck_130	0.909762	ngvck_170	0.801691
ngvck_051	0.937296	ngvck_091	0.83877	ngvck_131	0.826622	ngvck_171	0.826074
ngvck_052	0.946555	ngvck_092	0.779999	ngvck_132	0.95476	ngvck_172	0.853318
ngvck_053	0.985901	ngvck_093	0.811859	ngvck_133	0.851179	ngvck_173	0.824691
ngvck_054	0.97076	ngvck_094	0.764465	ngvck_134	0.903836	ngvck_174	**0.647984**
ngvck_055	0.976043	ngvck_095	0.757465	ngvck_135	0.763869	ngvck_175	0.829205
ngvck_056	1.019535	ngvck_096	0.874001	ngvck_136	0.901318	ngvck_176	0.942609
ngvck_057	0.837554	ngvck_097	0.868151	ngvck_137	0.864851	ngvck_177	0.932823
ngvck_058	0.896579	ngvck_098	0.843217	ngvck_138	0.848179	ngvck_178	0.811958
ngvck_059	1.021797	ngvck_099	0.838819	ngvck_139	0.780537	ngvck_179	0.811343
ngvck_060	0.906058	ngvck_100	0.833035	ngvck_140	0.817435	ngvck_180	0.972538
ngvck_061	0.848395	ngvck_101	0.888284	ngvck_141	0.798006	ngvck_181	0.844325
ngvck_062	0.851172	ngvck_102	0.807025	ngvck_142	0.876068	ngvck_182	0.911726
ngvck_063	0.840138	ngvck_103	0.834233	ngvck_143	0.806389	ngvck_183	0.86521
ngvck_064	0.873343	ngvck_104	0.859233	ngvck_144	0.859658	ngvck_184	0.809428
ngvck_065	0.908832	ngvck_105	0.823624	ngvck_145	0.801252	ngvck_185	0.898132
ngvck_066	0.798149	ngvck_106	0.748523	ngvck_146	0.832969	ngvck_186	0.849825
ngvck_067	0.865831	ngvck_107	0.772242	ngvck_147	0.854851	ngvck_187	0.866119
ngvck_068	0.86319	ngvck_108	0.857217	ngvck_148	0.84847	ngvck_188	0.929229
ngvck_069	0.822837	ngvck_109	0.925482	ngvck_149	0.827902	ngvck_189	0.909698
ngvck_070	0.79258	ngvck_110	0.927198	ngvck_150	0.763924	ngvck_190	0.875517
ngvck_071	0.791387	ngvck_111	0.818228	ngvck_151	0.792034	ngvck_191	0.787677
ngvck_072	0.753153	ngvck_112	0.876711	ngvck_152	0.851721	ngvck_192	0.819372
ngvck_073	0.700216	ngvck_113	0.763812	ngvck_153	0.838518	ngvck_193	0.859047
ngvck_074	0.856811	ngvck_114	0.851522	ngvck_154	0.842713	ngvck_194	0.911266
ngvck_075	0.810692	ngvck_115	0.802874	ngvck_155	0.976208	ngvck_195	0.901283
ngvck_076	0.759762	ngvck_116	0.783519	ngvck_156	0.7397	ngvck_196	0.778578
ngvck_077	0.839966	ngvck_117	0.911759	ngvck_157	0.85456	ngvck_197	0.815285
ngvck_078	0.913255	ngvck_118	0.752942	ngvck_158	0.761636	ngvck_198	0.807637
ngvck_079	0.895487	ngvck_119	0.937536	ngvck_159	0.826476	ngvck_199	0.88342
ngvck_080	0.819754	ngvck_120	0.765434	ngvck_160	0.904737	ngvck_200	0.931357

Figure C-3: Graphical representation of Virus and Non-Virus Scores using ngvck_ pp_group20_03 model

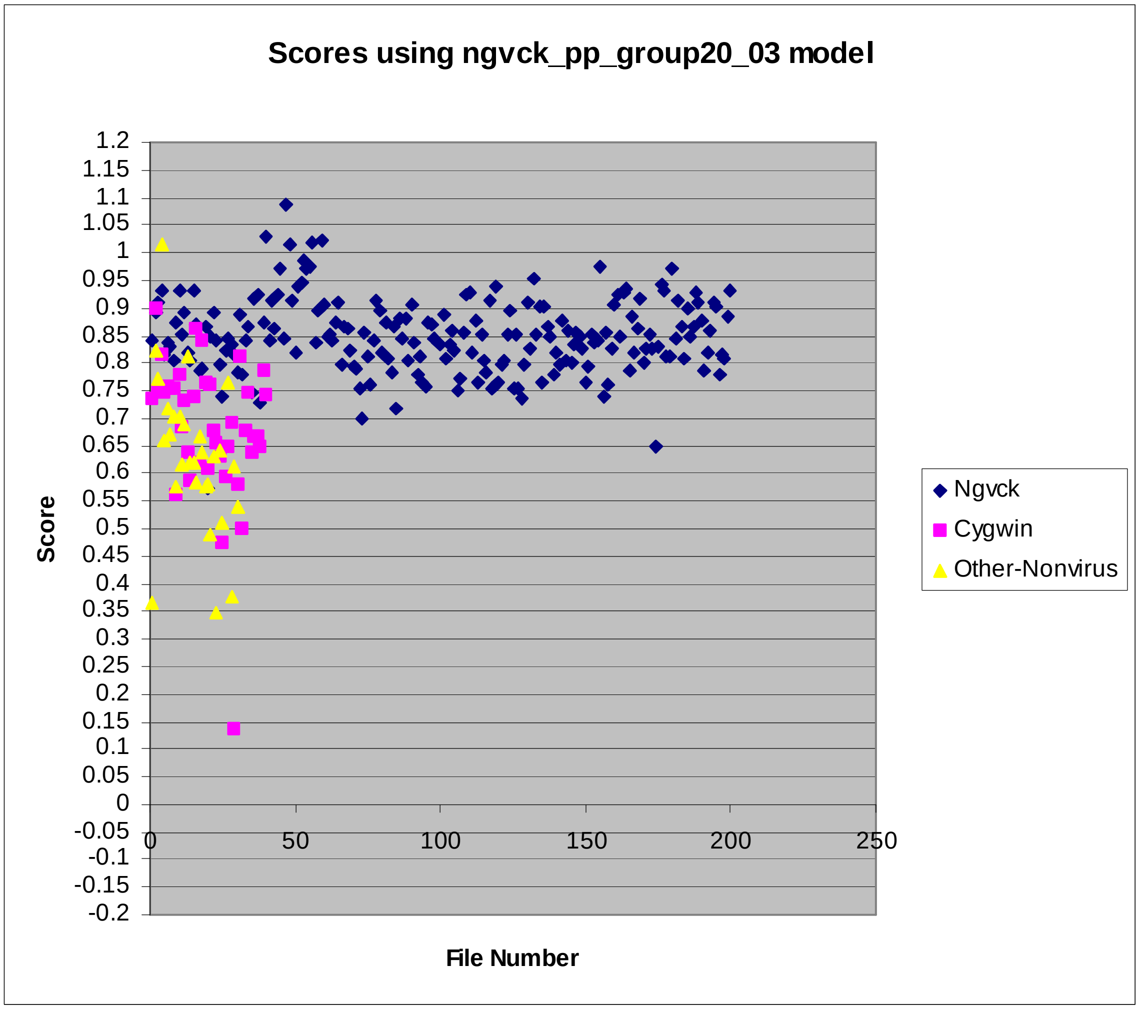


Table C-4.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_04 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.850841	cygwin_01	0.550491	nonvirus_01	0.182317
ngvck_002	0.789578	cygwin_02	**0.705151**	nonvirus_02	0.4411
ngvck_003	0.929644	cygwin_03	0.493297	nonvirus_03	0.368082
ngvck_004	0.757559	cygwin_04	0.593455	nonvirus_04	**0.686137**
ngvck_005	0.811504	cygwin_05	0.522902	nonvirus_05	0.335471
ngvck_006	0.856675	cygwin_06	0.51482	nonvirus_06	0.37703
ngvck_007	0.821737	cygwin_07	0.508263	nonvirus_07	0.358807
ngvck_008	0.793246	cygwin_08	0.542356	nonvirus_08	0.380057
ngvck_009	0.840646	cygwin_09	0.408046	nonvirus_09	0.294592
ngvck_010	0.838195	cygwin_10	0.590317	nonvirus_10	0.364261
ngvck_011	0.807864	cygwin_11	0.481398	nonvirus_11	0.539318
ngvck_012	0.814475	cygwin_12	0.538339	nonvirus_12	0.344576
ngvck_013	0.85686	cygwin_13	0.455474	nonvirus_13	0.401636
ngvck_014	0.687016	cygwin_14	0.397333	nonvirus_14	0.406111
ngvck_015	0.839257	cygwin_15	0.599099	nonvirus_15	0.426938
ngvck_016	0.901444	cygwin_16	0.65914	nonvirus_16	0.340582
ngvck_017	0.841206	cygwin_17	0.489615	nonvirus_17	0.351213
ngvck_018	0.73845	cygwin_18	0.614919	nonvirus_18	0.386637
ngvck_019	0.863958	cygwin_19	0.499818	nonvirus_19	0.284266
ngvck_020	**0.567517**	cygwin_20	0.442372	nonvirus_20	0.308129
ngvck_021	0.873405	cygwin_21	0.593168	nonvirus_21	0.267181
ngvck_022	0.726899	cygwin_22	0.492374	nonvirus_22	0.428343
ngvck_023	0.806073	cygwin_23	0.489134	nonvirus_23	0.12885
ngvck_024	0.816556	cygwin_24	0.478008	nonvirus_24	0.446545
ngvck_025	0.818126	cygwin_25	0.340155	nonvirus_25	0.289537
ngvck_026	0.864861	cygwin_26	0.455283	nonvirus_26	-2.922211
ngvck_027	0.869858	cygwin_27	0.486133	nonvirus_27	0.456843
ngvck_028	0.805993	cygwin_28	0.560022	nonvirus_28	0.298224
ngvck_029	0.830942	cygwin_29	0.113415	nonvirus_29	0.493696
ngvck_030	0.7618	cygwin_30	0.4258	nonvirus_30	0.391814
ngvck_031	0.832774	cygwin_31	**0.675477**		
ngvck_032	0.801001	cygwin_32	0.469703		
ngvck_033	0.87189	cygwin_33	0.447418		
ngvck_034	0.789558	cygwin_34	0.574049		
ngvck_035	0.767798	cygwin_35	0.462666		
ngvck_036	0.805831	cygwin_36	0.540654		
ngvck_037	0.824222	cygwin_37	0.549647		
ngvck_038	0.741865	cygwin_38	0.479169		
ngvck_039	0.867498	cygwin_39	0.550915		
ngvck_040	0.83168	cygwin_40	0.586272		

Table C-4.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_04 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.757298	ngvck_081	0.875376	ngvck_121	0.867213	ngvck_161	0.858928
ngvck_042	0.868983	ngvck_082	0.789727	ngvck_122	0.879113	ngvck_162	0.874478
ngvck_043	0.835482	ngvck_083	0.78406	ngvck_123	0.803316	ngvck_163	0.83829
ngvck_044	0.772492	ngvck_084	0.848924	ngvck_124	0.804353	ngvck_164	0.852214
ngvck_045	0.808767	ngvck_085	0.765512	ngvck_125	0.750825	ngvck_165	0.688109
ngvck_046	0.852955	ngvck_086	0.847363	ngvck_126	0.929491	ngvck_166	0.902067
ngvck_047	0.858213	ngvck_087	0.787948	ngvck_127	0.793311	ngvck_167	0.815573
ngvck_048	0.840456	ngvck_088	0.799455	ngvck_128	0.757188	ngvck_168	0.762217
ngvck_049	0.892527	ngvck_089	0.814521	ngvck_129	0.714807	ngvck_169	0.862011
ngvck_050	0.756263	ngvck_090	0.80263	ngvck_130	0.858053	ngvck_170	0.777249
ngvck_051	0.722567	ngvck_091	0.867838	ngvck_131	0.833891	ngvck_171	0.826157
ngvck_052	0.757073	ngvck_092	0.787624	ngvck_132	0.865877	ngvck_172	0.892451
ngvck_053	0.924156	ngvck_093	0.793503	ngvck_133	0.78726	ngvck_173	0.768491
ngvck_054	0.824019	ngvck_094	0.810422	ngvck_134	0.827229	ngvck_174	**0.666343**
ngvck_055	0.867228	ngvck_095	0.801993	ngvck_135	0.799894	ngvck_175	0.778774
ngvck_056	0.844169	ngvck_096	0.819189	ngvck_136	0.826665	ngvck_176	0.877213
ngvck_057	0.673355	ngvck_097	0.880471	ngvck_137	0.894569	ngvck_177	0.82546
ngvck_058	0.82388	ngvck_098	0.885093	ngvck_138	0.815387	ngvck_178	0.691058
ngvck_059	0.839189	ngvck_099	0.830922	ngvck_139	0.817584	ngvck_179	0.796368
ngvck_060	1.012009	ngvck_100	0.771427	ngvck_140	0.882078	ngvck_180	0.853128
ngvck_061	0.916303	ngvck_101	0.807563	ngvck_141	0.754774	ngvck_181	0.831712
ngvck_062	0.866677	ngvck_102	0.796399	ngvck_142	0.806499	ngvck_182	0.815922
ngvck_063	0.894036	ngvck_103	0.878371	ngvck_143	0.791995	ngvck_183	0.886166
ngvck_064	0.991646	ngvck_104	0.87129	ngvck_144	0.776753	ngvck_184	0.796525
ngvck_065	0.86211	ngvck_105	0.793809	ngvck_145	0.80383	ngvck_185	0.930065
ngvck_066	0.739855	ngvck_106	0.770843	ngvck_146	0.83658	ngvck_186	0.860386
ngvck_067	0.879567	ngvck_107	0.867642	ngvck_147	0.852187	ngvck_187	0.771396
ngvck_068	0.883107	ngvck_108	0.843888	ngvck_148	0.876262	ngvck_188	0.809346
ngvck_069	0.886513	ngvck_109	0.878378	ngvck_149	0.86039	ngvck_189	0.826969
ngvck_070	0.85795	ngvck_110	0.848251	ngvck_150	0.790311	ngvck_190	0.738429
ngvck_071	0.858721	ngvck_111	0.827482	ngvck_151	0.834198	ngvck_191	0.806646
ngvck_072	0.844036	ngvck_112	0.831778	ngvck_152	0.863463	ngvck_192	0.826995
ngvck_073	0.736049	ngvck_113	0.82623	ngvck_153	0.861613	ngvck_193	0.909475
ngvck_074	0.978836	ngvck_114	0.817299	ngvck_154	0.875987	ngvck_194	0.833949
ngvck_075	0.945607	ngvck_115	0.811942	ngvck_155	0.907116	ngvck_195	0.802299
ngvck_076	0.853917	ngvck_116	0.849045	ngvck_156	0.790272	ngvck_196	0.79544
ngvck_077	0.926047	ngvck_117	0.852727	ngvck_157	0.910911	ngvck_197	0.853541
ngvck_078	1.048576	ngvck_118	0.834679	ngvck_158	0.797436	ngvck_198	0.821007
ngvck_079	0.907694	ngvck_119	0.772825	ngvck_159	0.840191	ngvck_199	0.805343
ngvck_080	0.7952	ngvck_120	0.811841	ngvck_160	0.834498	ngvck_200	0.852154

Figure C-4: Graphical representation of Virus and Non-Virus Scores using ngvck_ pp_group20_04 model

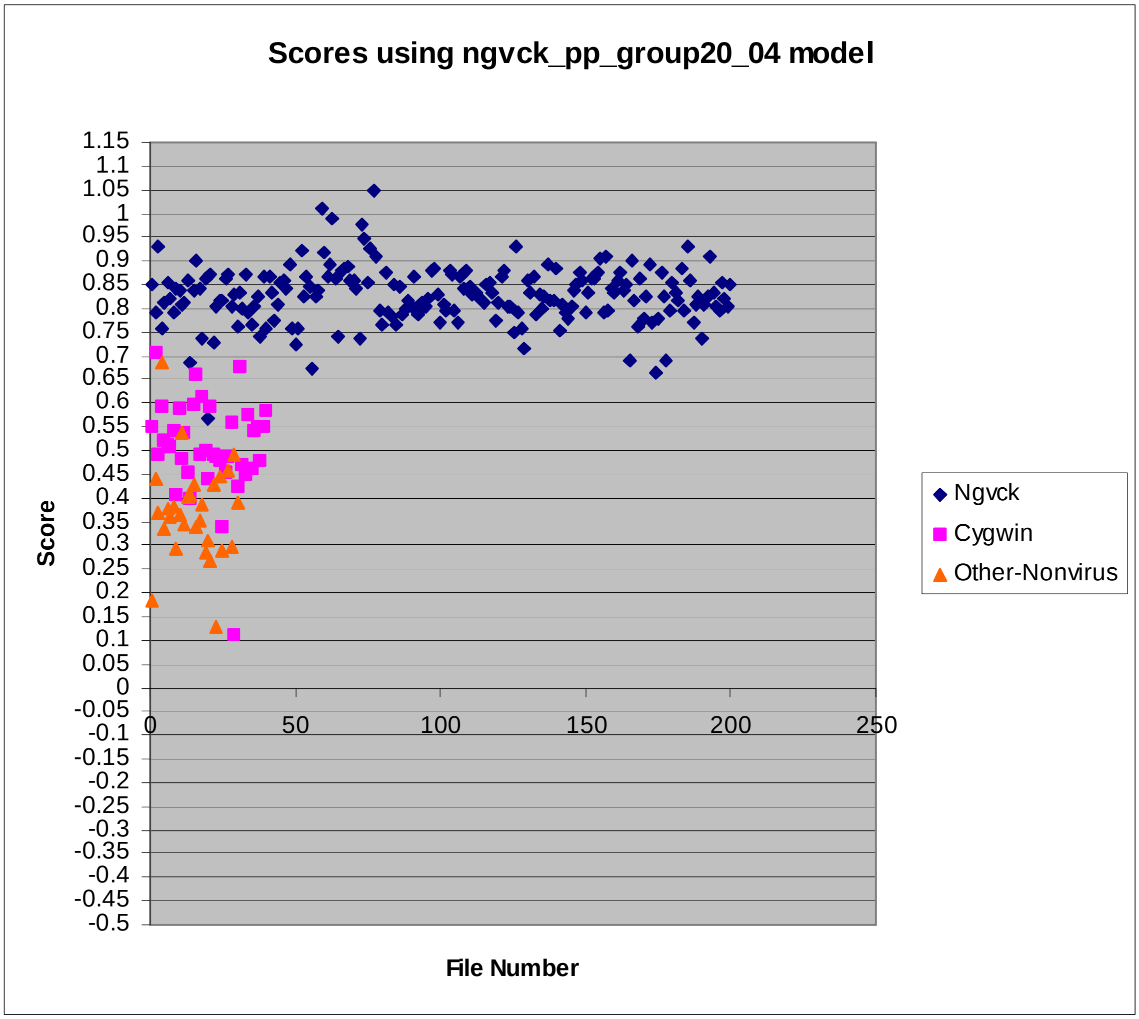


Table C-5.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_05 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.860312	cygwin_01	0.65142	nonvirus_01	0.3098
ngvck_002	0.795237	cygwin_02	**0.837838**	nonvirus_02	0.548429
ngvck_003	0.886658	cygwin_03	0.62421	nonvirus_03	0.461659
ngvck_004	0.802831	cygwin_04	**0.712136**	nonvirus_04	**0.792522**
ngvck_005	0.81137	cygwin_05	0.598098	nonvirus_05	0.403953
ngvck_006	0.839508	cygwin_06	0.602079	nonvirus_06	0.454446
ngvck_007	0.849245	cygwin_07	0.595102	nonvirus_07	0.408925
ngvck_008	0.795148	cygwin_08	0.619307	nonvirus_08	0.451406
ngvck_009	0.834733	cygwin_09	0.522162	nonvirus_09	0.403035
ngvck_010	0.84878	cygwin_10	**0.766615**	nonvirus_10	0.43616
ngvck_011	0.808141	cygwin_11	0.625508	nonvirus_11	0.47951
ngvck_012	0.835057	cygwin_12	0.633505	nonvirus_12	0.422449
ngvck_013	0.829593	cygwin_13	0.545201	nonvirus_13	0.559038
ngvck_014	0.713438	cygwin_14	0.511541	nonvirus_14	0.38252
ngvck_015	0.842663	cygwin_15	0.690079	nonvirus_15	0.405813
ngvck_016	0.873962	cygwin_16	**0.795933**	nonvirus_16	0.454967
ngvck_017	0.824749	cygwin_17	0.573789	nonvirus_17	0.414138
ngvck_018	0.738526	cygwin_18	**0.747769**	nonvirus_18	0.491692
ngvck_019	0.834267	cygwin_19	0.645287	nonvirus_19	0.353092
ngvck_020	**0.5836**71	cygwin_20	0.545467	nonvirus_20	0.348321
ngvck_021	0.828935	cygwin_21	**0.707892**	nonvirus_21	0.323091
ngvck_022	0.758629	cygwin_22	0.673296	nonvirus_22	0.456602
ngvck_023	0.819315	cygwin_23	0.506957	nonvirus_23	0.212399
ngvck_024	0.793526	cygwin_24	0.565108	nonvirus_24	0.39308
ngvck_025	0.766219	cygwin_25	0.553429	nonvirus_25	0.355802
ngvck_026	0.847906	cygwin_26	0.569064	nonvirus_26	-2.666368
ngvck_027	0.840946	cygwin_27	0.593251	nonvirus_27	0.598836
ngvck_028	0.833153	cygwin_28	0.642484	nonvirus_28	0.353773
ngvck_029	0.822582	cygwin_29	0.124127	nonvirus_29	0.434932
ngvck_030	0.752658	cygwin_30	0.573054	nonvirus_30	0.351344
ngvck_031	0.832907	cygwin_31	**0.784436**		
ngvck_032	0.760973	cygwin_32	0.502194		
ngvck_033	0.895305	cygwin_33	0.593266		
ngvck_034	0.799552	cygwin_34	**0.732766**		
ngvck_035	0.756854	cygwin_35	0.626555		
ngvck_036	0.841632	cygwin_36	0.681297		
ngvck_037	0.838682	cygwin_37	0.643173		
ngvck_038	0.758617	cygwin_38	0.619963		
ngvck_039	0.862616	cygwin_39	**0.690918**		
ngvck_040	0.820418	cygwin_40	**0.704758**		

Table C-5.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_05 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.789268	ngvck_081	0.949979	ngvck_121	0.786711	ngvck_161	0.848393
ngvck_042	0.877905	ngvck_082	0.839963	ngvck_122	0.876532	ngvck_162	0.858623
ngvck_043	0.871348	ngvck_083	0.829506	ngvck_123	0.775447	ngvck_163	0.82717
ngvck_044	0.791164	ngvck_084	0.9695	ngvck_124	0.825083	ngvck_164	0.861848
ngvck_045	0.847984	ngvck_085	0.788568	ngvck_125	0.741749	ngvck_165	0.731031
ngvck_046	0.853752	ngvck_086	0.914138	ngvck_126	0.91602	ngvck_166	0.927098
ngvck_047	0.882946	ngvck_087	0.843457	ngvck_127	0.773638	ngvck_167	0.820945
ngvck_048	0.833787	ngvck_088	0.87577	ngvck_128	0.771198	ngvck_168	0.768382
ngvck_049	0.914435	ngvck_089	0.903105	ngvck_129	0.75079	ngvck_169	0.847036
ngvck_050	0.738183	ngvck_090	0.896721	ngvck_130	0.855277	ngvck_170	0.773147
ngvck_051	0.730682	ngvck_091	0.885369	ngvck_131	0.846253	ngvck_171	0.857833
ngvck_052	0.78623	ngvck_092	0.822293	ngvck_132	0.874133	ngvck_172	0.880044
ngvck_053	0.85688	ngvck_093	0.848912	ngvck_133	0.776646	ngvck_173	0.78647
ngvck_054	0.821237	ngvck_094	0.842295	ngvck_134	0.835827	ngvck_174	**0.666196**
ngvck_055	0.875092	ngvck_095	0.854048	ngvck_135	0.781461	ngvck_175	0.785309
ngvck_056	0.885147	ngvck_096	0.827899	ngvck_136	0.841166	ngvck_176	0.88969
ngvck_057	0.740422	ngvck_097	1.012343	ngvck_137	0.868623	ngvck_177	0.837363
ngvck_058	0.848453	ngvck_098	0.982843	ngvck_138	0.832373	ngvck_178	0.690419
ngvck_059	0.868572	ngvck_099	0.898596	ngvck_139	0.793071	ngvck_179	0.789939
ngvck_060	0.889875	ngvck_100	0.798248	ngvck_140	0.876543	ngvck_180	0.909156
ngvck_061	0.847608	ngvck_101	0.828098	ngvck_141	0.780222	ngvck_181	0.858647
ngvck_062	0.839004	ngvck_102	0.823811	ngvck_142	0.803766	ngvck_182	0.846342
ngvck_063	0.86194	ngvck_103	0.823456	ngvck_143	0.790492	ngvck_183	0.89333
ngvck_064	0.892939	ngvck_104	0.884113	ngvck_144	0.767038	ngvck_184	0.781806
ngvck_065	0.849876	ngvck_105	0.835814	ngvck_145	0.82432	ngvck_185	0.91926
ngvck_066	0.721508	ngvck_106	0.79369	ngvck_146	0.865609	ngvck_186	0.858923
ngvck_067	0.797511	ngvck_107	0.812982	ngvck_147	0.891862	ngvck_187	0.800423
ngvck_068	0.868227	ngvck_108	0.933249	ngvck_148	0.879315	ngvck_188	0.80888
ngvck_069	0.818536	ngvck_109	0.886441	ngvck_149	0.828714	ngvck_189	0.792674
ngvck_070	0.805478	ngvck_110	0.889064	ngvck_150	0.811242	ngvck_190	0.755954
ngvck_071	0.78848	ngvck_111	0.862638	ngvck_151	0.774718	ngvck_191	0.769365
ngvck_072	0.802295	ngvck_112	0.847809	ngvck_152	0.858661	ngvck_192	0.832916
ngvck_073	0.708494	ngvck_113	0.787307	ngvck_153	0.870033	ngvck_193	0.904256
ngvck_074	0.894591	ngvck_114	0.874534	ngvck_154	0.882416	ngvck_194	0.835516
ngvck_075	0.820317	ngvck_115	0.82549	ngvck_155	0.918738	ngvck_195	0.812253
ngvck_076	0.8137	ngvck_116	0.802846	ngvck_156	0.746016	ngvck_196	0.772325
ngvck_077	0.860255	ngvck_117	0.848065	ngvck_157	0.872844	ngvck_197	0.838026
ngvck_078	0.912617	ngvck_118	0.814558	ngvck_158	0.777988	ngvck_198	0.811225
ngvck_079	0.850906	ngvck_119	0.853697	ngvck_159	0.877765	ngvck_199	0.833222
ngvck_080	0.953381	ngvck_120	0.803989	ngvck_160	0.86788	ngvck_200	0.865753

Figure C-5: Graphical representation of Virus and Non-Virus Scores using ngvck_ pp_group20_05 model

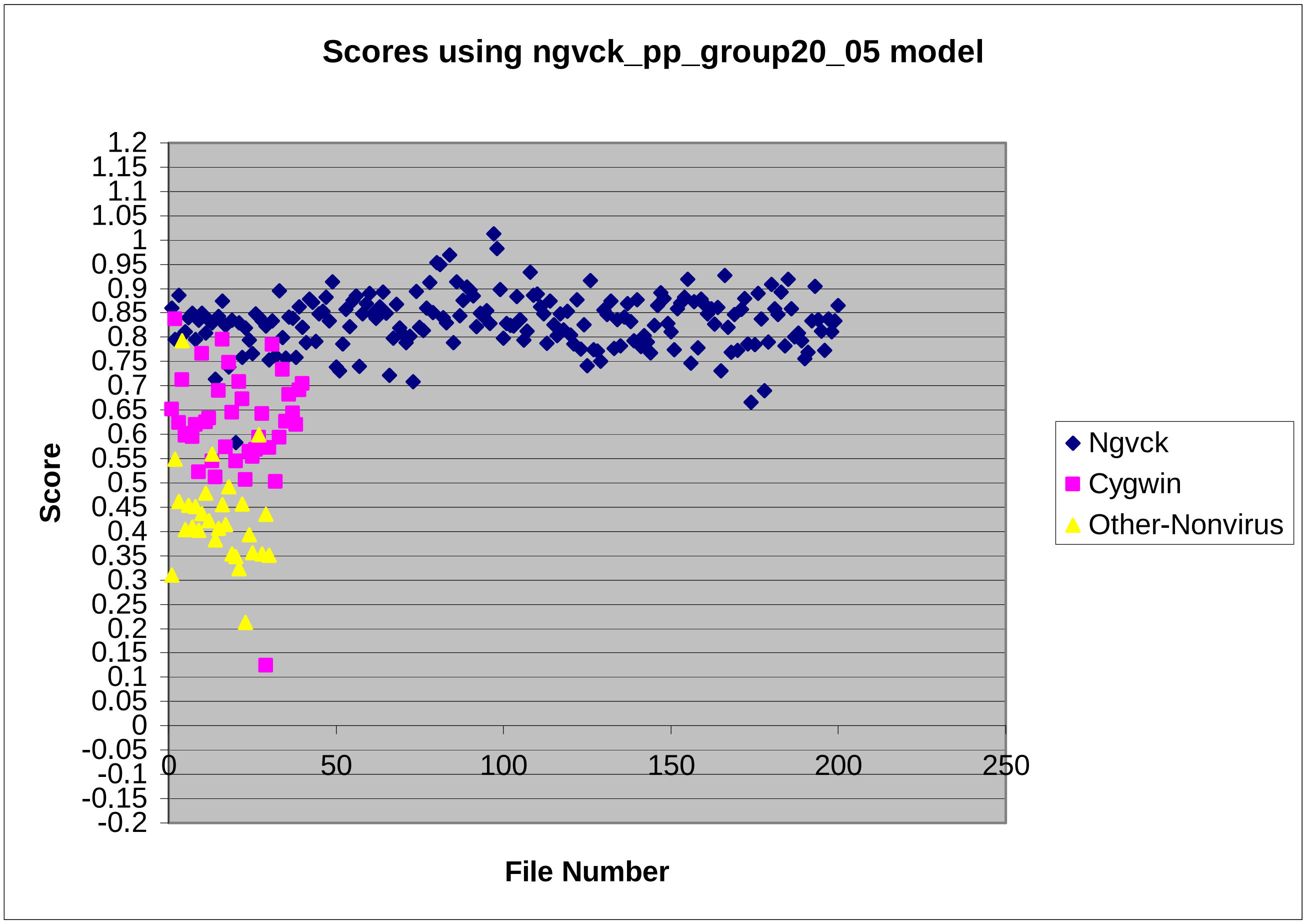


Table C-6.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_06 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.857348	cygwin_01	0.665388	nonvirus_01	0.344736
ngvck_002	0.830591	cygwin_02	**0.835151**	nonvirus_02	**0.708166**
ngvck_003	0.886336	cygwin_03	0.645472	nonvirus_03	0.669798
ngvck_004	0.780231	cygwin_04	**0.729581**	nonvirus_04	**0.885802**
ngvck_005	0.835773	cygwin_05	**0.704802**	nonvirus_05	0.571462
ngvck_006	0.844064	cygwin_06	**0.698572**	nonvirus_06	0.631871
ngvck_007	0.82485	cygwin_07	0.686196	nonvirus_07	0.596628
ngvck_008	0.810281	cygwin_08	**0.719053**	nonvirus_08	0.621427
ngvck_009	0.856378	cygwin_09	0.53585	nonvirus_09	0.481649
ngvck_010	0.892242	cygwin_10	**0.725013**	nonvirus_10	0.598287
ngvck_011	0.776092	cygwin_11	0.637629	nonvirus_11	0.510082
ngvck_012	0.90029	cygwin_12	0.650058	nonvirus_12	0.567451
ngvck_013	0.834927	cygwin_13	0.550787	nonvirus_13	**0.69617**
ngvck_014	0.716053	cygwin_14	0.560818	nonvirus_14	0.51911
ngvck_015	0.900905	cygwin_15	**0.713838**	nonvirus_15	0.530152
ngvck_016	0.870551	cygwin_16	**0.790463**	nonvirus_16	0.526742
ngvck_017	0.781267	cygwin_17	0.589251	nonvirus_17	0.591833
ngvck_018	0.711871	cygwin_18	**0.744832**	nonvirus_18	0.507001
ngvck_019	0.87085	cygwin_19	0.662689	nonvirus_19	0.477605
ngvck_020	**0.565175**	cygwin_20	0.563861	nonvirus_20	0.518567
ngvck_021	0.815945	cygwin_21	**0.711819**	nonvirus_21	0.450247
ngvck_022	0.792282	cygwin_22	0.645836	nonvirus_22	0.519793
ngvck_023	0.824134	cygwin_23	0.523773	nonvirus_23	0.284339
ngvck_024	0.825402	cygwin_24	0.598631	nonvirus_24	0.513377
ngvck_025	0.780611	cygwin_25	0.454176	nonvirus_25	0.450295
ngvck_026	0.857428	cygwin_26	0.606585	nonvirus_26	-2.84037
ngvck_027	0.841704	cygwin_27	0.618683	nonvirus_27	0.637178
ngvck_028	0.842293	cygwin_28	0.666782	nonvirus_28	0.360564
ngvck_029	0.812517	cygwin_29	0.101304	nonvirus_29	0.469775
ngvck_030	0.77842	cygwin_30	0.560581	nonvirus_30	0.504919
ngvck_031	0.840315	cygwin_31	**0.793235**		
ngvck_032	0.738386	cygwin_32	0.529049		
ngvck_033	0.878783	cygwin_33	0.576481		
ngvck_034	0.847432	cygwin_34	**0.711174**		
ngvck_035	0.755223	cygwin_35	0.633463		
ngvck_036	0.844135	cygwin_36	0.677069		
ngvck_037	0.875078	cygwin_37	0.6667		
ngvck_038	0.764438	cygwin_38	0.568887		
ngvck_039	0.848934	cygwin_39	**0.729031**		
ngvck_040	0.878674	cygwin_40	**0.715616**		

Table C-6.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_06 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.787965	ngvck_081	0.919251	ngvck_121	0.773551	ngvck_161	0.886497
ngvck_042	0.874135	ngvck_082	0.842519	ngvck_122	0.850995	ngvck_162	0.865237
ngvck_043	0.823588	ngvck_083	0.790639	ngvck_123	0.836089	ngvck_163	0.903766
ngvck_044	0.86921	ngvck_084	0.891821	ngvck_124	0.875467	ngvck_164	0.871218
ngvck_045	0.827033	ngvck_085	0.791448	ngvck_125	0.740726	ngvck_165	0.693498
ngvck_046	0.824635	ngvck_086	0.899106	ngvck_126	0.919251	ngvck_166	0.932124
ngvck_047	0.899466	ngvck_087	0.837336	ngvck_127	0.779074	ngvck_167	0.835308
ngvck_048	0.864145	ngvck_088	0.861653	ngvck_128	0.822191	ngvck_168	0.821496
ngvck_049	0.881111	ngvck_089	0.797263	ngvck_129	0.755359	ngvck_169	0.896264
ngvck_050	0.744739	ngvck_090	0.837355	ngvck_130	0.916226	ngvck_170	0.771048
ngvck_051	0.780073	ngvck_091	0.847147	ngvck_131	0.891337	ngvck_171	0.860619
ngvck_052	0.797362	ngvck_092	0.790581	ngvck_132	0.904379	ngvck_172	0.868688
ngvck_053	0.855975	ngvck_093	0.828515	ngvck_133	0.840713	ngvck_173	0.828928
ngvck_054	0.869458	ngvck_094	0.798159	ngvck_134	0.901	ngvck_174	**0.658104**
ngvck_055	0.906181	ngvck_095	0.806778	ngvck_135	0.780507	ngvck_175	0.800254
ngvck_056	0.906896	ngvck_096	0.855954	ngvck_136	0.889069	ngvck_176	0.911872
ngvck_057	0.7225	ngvck_097	0.909905	ngvck_137	0.874965	ngvck_177	0.864514
ngvck_058	0.833423	ngvck_098	0.883154	ngvck_138	0.868311	ngvck_178	0.715396
ngvck_059	0.884655	ngvck_099	0.875773	ngvck_139	0.788287	ngvck_179	0.789318
ngvck_060	0.858085	ngvck_100	0.877474	ngvck_140	0.846438	ngvck_180	0.915722
ngvck_061	0.866041	ngvck_101	0.941228	ngvck_141	0.795904	ngvck_181	0.846975
ngvck_062	0.851616	ngvck_102	0.863846	ngvck_142	0.861356	ngvck_182	0.87459
ngvck_063	0.867943	ngvck_103	0.856345	ngvck_143	0.811352	ngvck_183	0.864454
ngvck_064	0.869142	ngvck_104	0.897436	ngvck_144	0.826857	ngvck_184	0.804237
ngvck_065	0.910372	ngvck_105	0.875443	ngvck_145	0.812919	ngvck_185	0.909293
ngvck_066	0.741648	ngvck_106	0.864577	ngvck_146	0.847005	ngvck_186	0.860541
ngvck_067	0.879336	ngvck_107	0.904318	ngvck_147	0.889716	ngvck_187	0.821049
ngvck_068	0.867364	ngvck_108	1.005835	ngvck_148	0.891082	ngvck_188	0.86864
ngvck_069	0.806277	ngvck_109	1.014893	ngvck_149	0.835956	ngvck_189	0.888359
ngvck_070	0.820465	ngvck_110	1.026811	ngvck_150	0.786074	ngvck_190	0.792541
ngvck_071	0.80174	ngvck_111	0.919397	ngvck_151	0.776276	ngvck_191	0.804301
ngvck_072	0.783952	ngvck_112	0.913977	ngvck_152	0.851057	ngvck_192	0.795243
ngvck_073	0.726149	ngvck_113	0.848819	ngvck_153	0.829074	ngvck_193	0.87558
ngvck_074	0.880718	ngvck_114	0.905502	ngvck_154	0.844509	ngvck_194	0.863965
ngvck_075	0.856148	ngvck_115	0.949807	ngvck_155	0.974118	ngvck_195	0.842996
ngvck_076	0.779281	ngvck_116	0.844209	ngvck_156	0.769981	ngvck_196	0.784674
ngvck_077	0.875151	ngvck_117	0.958554	ngvck_157	0.899375	ngvck_197	0.86301
ngvck_078	0.908855	ngvck_118	0.869516	ngvck_158	0.756111	ngvck_198	0.772871
ngvck_079	0.895629	ngvck_119	0.982696	ngvck_159	0.848037	ngvck_199	0.817451
ngvck_080	0.845306	ngvck_120	0.790235	ngvck_160	0.883175	ngvck_200	0.928114

Figure C-6: Graphical representation of Virus and Non-Virus Scores using ngvck_pp_group20_06 model

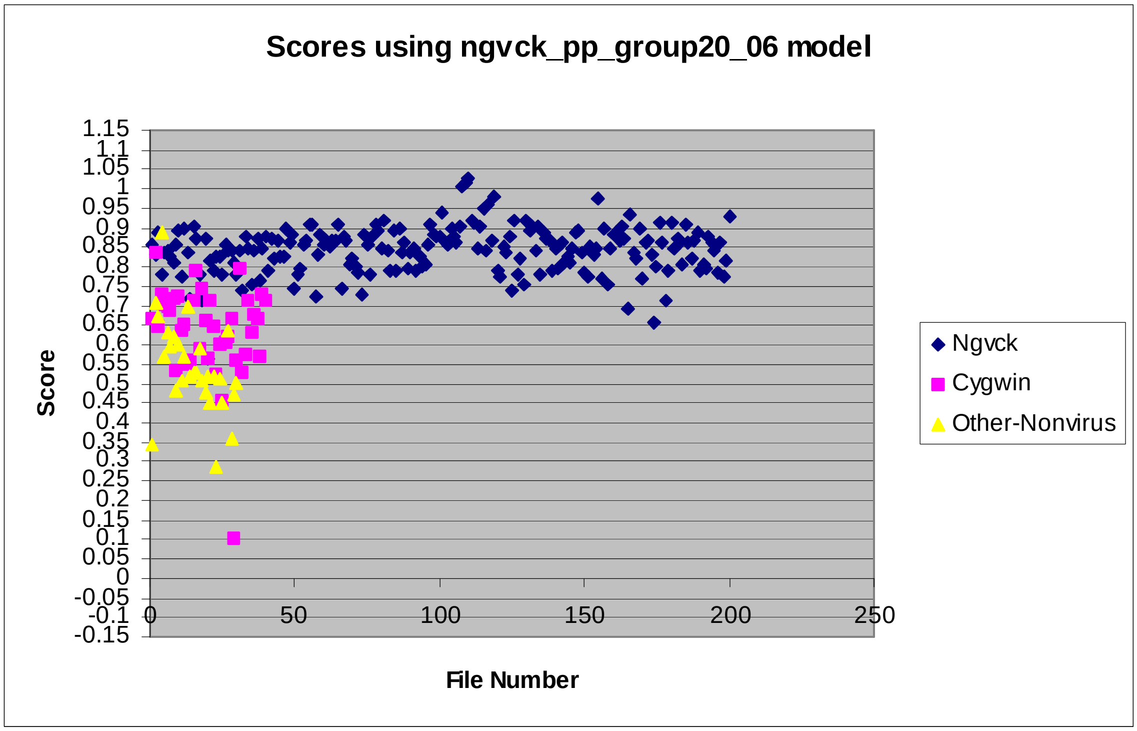


Table C-7.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_07 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.865543	cygwin_01	0.630754	nonvirus_01	0.362882
ngvck_002	0.814869	cygwin_02	**0.741246**	nonvirus_02	0.669439
ngvck_003	0.866258	cygwin_03	0.585521	nonvirus_03	0.551523
ngvck_004	0.800929	cygwin_04	0.652538	nonvirus_04	**0.777995**
ngvck_005	0.793267	cygwin_05	0.637812	nonvirus_05	0.536817
ngvck_006	0.82085	cygwin_06	0.636536	nonvirus_06	0.583833
ngvck_007	0.792142	cygwin_07	0.632793	nonvirus_07	0.548286
ngvck_008	0.770744	cygwin_08	0.649109	nonvirus_08	0.573075
ngvck_009	0.813397	cygwin_09	0.503038	nonvirus_09	0.489976
ngvck_010	0.88659	cygwin_10	0.627092	nonvirus_10	0.545885
ngvck_011	0.777947	cygwin_11	0.582739	nonvirus_11	0.494611
ngvck_012	0.883008	cygwin_12	0.603015	nonvirus_12	0.536247
ngvck_013	0.800081	cygwin_13	0.518257	nonvirus_13	0.627552
ngvck_014	0.730695	cygwin_14	0.50358	nonvirus_14	0.48205
ngvck_015	0.858821	cygwin_15	0.647903	nonvirus_15	0.500026
ngvck_016	0.874824	cygwin_16	**0.721082**	nonvirus_16	0.507018
ngvck_017	0.804879	cygwin_17	0.534949	nonvirus_17	0.560958
ngvck_018	0.695072	cygwin_18	**0.674909**	nonvirus_18	0.524389
ngvck_019	0.853693	cygwin_19	0.607236	nonvirus_19	0.48448
ngvck_020	**0.544857**	cygwin_20	0.528619	nonvirus_20	0.461801
ngvck_021	0.805895	cygwin_21	0.644642	nonvirus_21	0.421492
ngvck_022	0.780706	cygwin_22	0.60521	nonvirus_22	0.517388
ngvck_023	0.777135	cygwin_23	0.526276	nonvirus_23	0.301061
ngvck_024	0.79111	cygwin_24	0.540287	nonvirus_24	0.531684
ngvck_025	0.748803	cygwin_25	0.514545	nonvirus_25	0.482876
ngvck_026	0.848425	cygwin_26	0.517585	nonvirus_26	-2.901623
ngvck_027	0.81959	cygwin_27	0.540059	nonvirus_27	0.605216
ngvck_028	0.841042	cygwin_28	0.615562	nonvirus_28	0.32678
ngvck_029	0.773215	cygwin_29	0.082993	nonvirus_29	0.513358
ngvck_030	0.753541	cygwin_30	0.525204	nonvirus_30	0.484492
ngvck_031	0.880447	cygwin_31	**0.713515**		
ngvck_032	0.7555	cygwin_32	0.465451		
ngvck_033	0.875635	cygwin_33	0.544242		
ngvck_034	0.846532	cygwin_34	**0.663104**		
ngvck_035	0.750527	cygwin_35	0.580656		
ngvck_036	0.877516	cygwin_36	0.629654		
ngvck_037	0.837036	cygwin_37	0.600762		
ngvck_038	0.736628	cygwin_38	0.539433		
ngvck_039	0.840309	cygwin_39	**0.68919**		
ngvck_040	0.845677	cygwin_40	0.649597		

Table C-7.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_07 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.775355	ngvck_081	0.849711	ngvck_121	0.922586	ngvck_161	0.887096
ngvck_042	0.847094	ngvck_082	0.793204	ngvck_122	0.901097	ngvck_162	0.843595
ngvck_043	0.793334	ngvck_083	0.767332	ngvck_123	0.982504	ngvck_163	0.860205
ngvck_044	0.819569	ngvck_084	0.848144	ngvck_124	0.954673	ngvck_164	0.892938
ngvck_045	0.90113	ngvck_085	0.732159	ngvck_125	0.772622	ngvck_165	0.686417
ngvck_046	0.785737	ngvck_086	0.868704	ngvck_126	0.908331	ngvck_166	0.875288
ngvck_047	0.911622	ngvck_087	0.846198	ngvck_127	0.817109	ngvck_167	0.803674
ngvck_048	0.859318	ngvck_088	0.83673	ngvck_128	0.789825	ngvck_168	0.786548
ngvck_049	0.86345	ngvck_089	0.787165	ngvck_129	0.831343	ngvck_169	0.887587
ngvck_050	0.728942	ngvck_090	0.879452	ngvck_130	0.92762	ngvck_170	0.771826
ngvck_051	0.734492	ngvck_091	0.828947	ngvck_131	0.845319	ngvck_171	0.788454
ngvck_052	0.801667	ngvck_092	0.784641	ngvck_132	0.952769	ngvck_172	0.8577
ngvck_053	0.873191	ngvck_093	0.803368	ngvck_133	0.967479	ngvck_173	0.805117
ngvck_054	0.843231	ngvck_094	0.748867	ngvck_134	0.972339	ngvck_174	**0.654976**
ngvck_055	0.900406	ngvck_095	0.788409	ngvck_135	0.792553	ngvck_175	0.813911
ngvck_056	0.847508	ngvck_096	0.828265	ngvck_136	0.926932	ngvck_176	0.87807
ngvck_057	0.734386	ngvck_097	0.850101	ngvck_137	0.929807	ngvck_177	0.871574
ngvck_058	0.818712	ngvck_098	0.861416	ngvck_138	0.885438	ngvck_178	0.729826
ngvck_059	0.900183	ngvck_099	0.804601	ngvck_139	0.841805	ngvck_179	0.763426
ngvck_060	0.843617	ngvck_100	0.805193	ngvck_140	0.820611	ngvck_180	0.914973
ngvck_061	0.843202	ngvck_101	0.872581	ngvck_141	0.795471	ngvck_181	0.793061
ngvck_062	0.802549	ngvck_102	0.787962	ngvck_142	0.835331	ngvck_182	0.87821
ngvck_063	0.840131	ngvck_103	0.797834	ngvck_143	0.81389	ngvck_183	0.827427
ngvck_064	0.85059	ngvck_104	0.849879	ngvck_144	0.812214	ngvck_184	0.773326
ngvck_065	0.88008	ngvck_105	0.828476	ngvck_145	0.789994	ngvck_185	0.867622
ngvck_066	0.740655	ngvck_106	0.761005	ngvck_146	0.825814	ngvck_186	0.843866
ngvck_067	0.83688	ngvck_107	0.774156	ngvck_147	0.822584	ngvck_187	0.863482
ngvck_068	0.838369	ngvck_108	0.867225	ngvck_148	0.847318	ngvck_188	0.845432
ngvck_069	0.805139	ngvck_109	0.906238	ngvck_149	0.78061	ngvck_189	0.8818
ngvck_070	0.801444	ngvck_110	0.932246	ngvck_150	0.769145	ngvck_190	0.772034
ngvck_071	0.805832	ngvck_111	0.809172	ngvck_151	0.767872	ngvck_191	0.784859
ngvck_072	0.781726	ngvck_112	0.851759	ngvck_152	0.875141	ngvck_192	0.802599
ngvck_073	0.707756	ngvck_113	0.785005	ngvck_153	0.870604	ngvck_193	0.869056
ngvck_074	0.839369	ngvck_114	0.836121	ngvck_154	0.839702	ngvck_194	0.864655
ngvck_075	0.813256	ngvck_115	0.809142	ngvck_155	0.910136	ngvck_195	0.828821
ngvck_076	0.762005	ngvck_116	0.769565	ngvck_156	0.77269	ngvck_196	0.756132
ngvck_077	0.830247	ngvck_117	0.874432	ngvck_157	0.871158	ngvck_197	0.8267
ngvck_078	0.92745	ngvck_118	0.791796	ngvck_158	0.741797	ngvck_198	0.750452
ngvck_079	0.856045	ngvck_119	0.878888	ngvck_159	0.813947	ngvck_199	0.811506
ngvck_080	0.786227	ngvck_120	0.792326	ngvck_160	0.85236	ngvck_200	0.898386

Figure C-7: Graphical representation of Virus and Non-Virus Scores using ngvck_pp_group20_07 model

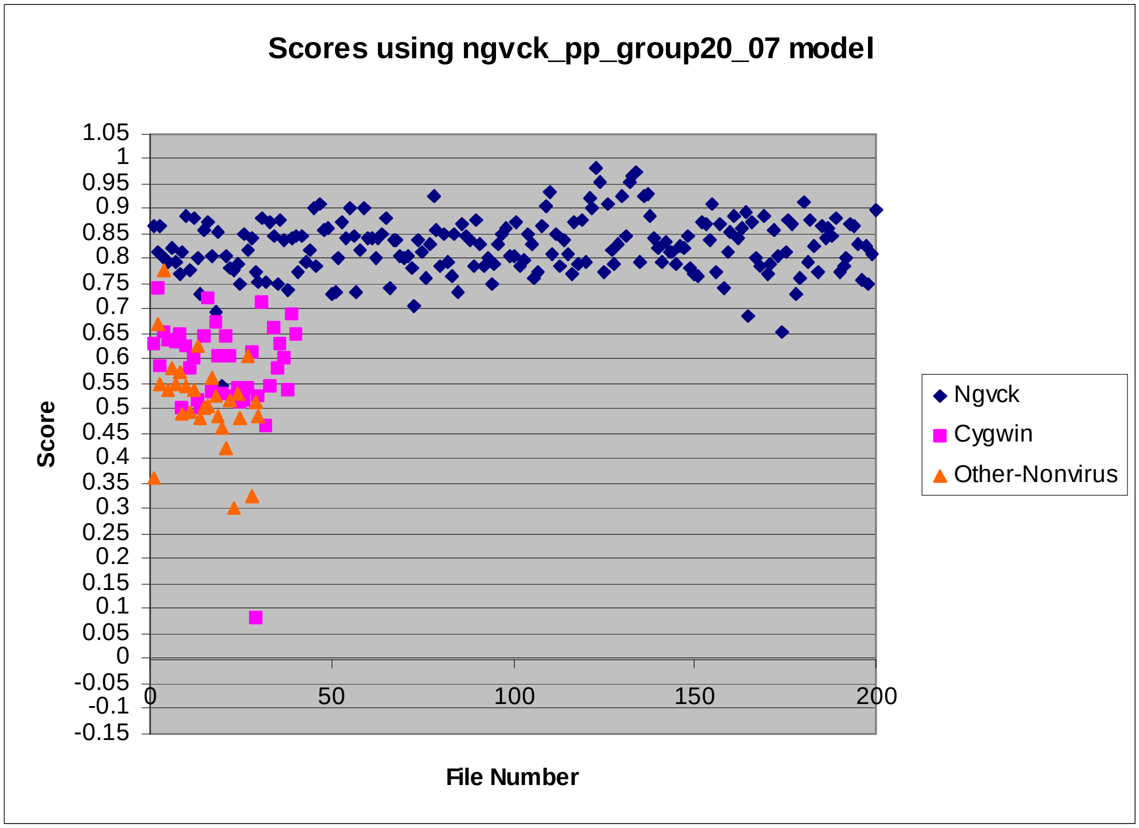


Table C-8.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_08 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.890141	cygwin_01	**0.691074**	nonvirus_01	0.404595
ngvck_002	0.777603	cygwin_02	**0.862468**	nonvirus_02	**0.801461**
ngvck_003	0.919734	cygwin_03	0.665714	nonvirus_03	**0.710558**
ngvck_004	0.745359	cygwin_04	**0.767715**	nonvirus_04	**0.973881**
ngvck_005	0.855405	cygwin_05	**0.700638**	nonvirus_05	0.644307
ngvck_006	0.879181	cygwin_06	**0.705133**	nonvirus_06	**0.699958**
ngvck_007	0.864211	cygwin_07	**0.700183**	nonvirus_07	0.662902
ngvck_008	0.811957	cygwin_08	**0.712667**	nonvirus_08	**0.691914**
ngvck_009	0.843758	cygwin_09	0.576687	nonvirus_09	0.502399
ngvck_010	0.835925	cygwin_10	**0.735554**	nonvirus_10	**0.680546**
ngvck_011	0.786572	cygwin_11	0.637736	nonvirus_11	0.558661
ngvck_012	0.834035	cygwin_12	0.677864	nonvirus_12	0.651092
ngvck_013	0.854765	cygwin_13	0.590888	nonvirus_13	**0.761642**
ngvck_014	0.68875	cygwin_14	0.548772	nonvirus_14	0.561532
ngvck_015	0.83759	cygwin_15	**0.715496**	nonvirus_15	0.569489
ngvck_016	0.948309	cygwin_16	**0.818346**	nonvirus_16	0.586006
ngvck_017	0.817287	cygwin_17	0.599846	nonvirus_17	0.677293
ngvck_018	0.755212	cygwin_18	**0.794147**	nonvirus_18	0.609735
ngvck_019	0.879627	cygwin_19	**0.712323**	nonvirus_19	0.575809
ngvck_020	**0.582577**	cygwin_20	0.571657	nonvirus_20	0.553802
ngvck_021	0.882238	cygwin_21	**0.723099**	nonvirus_21	0.492475
ngvck_022	0.711032	cygwin_22	**0.695721**	nonvirus_22	0.58752
ngvck_023	0.816493	cygwin_23	0.543123	nonvirus_23	0.323355
ngvck_024	0.862073	cygwin_24	0.602569	nonvirus_24	0.602793
ngvck_025	0.82815	cygwin_25	0.581748	nonvirus_25	0.568189
ngvck_026	0.884607	cygwin_26	0.59061	nonvirus_26	-2.847293
ngvck_027	0.912054	cygwin_27	0.624209	nonvirus_27	**0.727102**
ngvck_028	0.867979	cygwin_28	0.660262	nonvirus_28	0.34403
ngvck_029	0.831158	cygwin_29	0.097214	nonvirus_29	0.594118
ngvck_030	0.782319	cygwin_30	0.546242	nonvirus_30	0.537
ngvck_031	0.832386	cygwin_31	**0.798378**		
ngvck_032	0.808949	cygwin_32	0.488291		
ngvck_033	0.909359	cygwin_33	0.644436		
ngvck_034	0.792839	cygwin_34	**0.756893**		
ngvck_035	0.730825	cygwin_35	0.630445		
ngvck_036	0.841126	cygwin_36	**0.690675**		
ngvck_037	0.833045	cygwin_37	**0.689045**		
ngvck_038	0.768074	cygwin_38	0.637685		
ngvck_039	0.844888	cygwin_39	**0.736579**		
ngvck_040	0.826928	cygwin_40	**0.726946**		

Table C-8.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_08 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.805377	ngvck_081	0.915683	ngvck_121	0.833005	ngvck_161	0.849962
ngvck_042	0.896656	ngvck_082	0.831021	ngvck_122	0.880849	ngvck_162	0.87475
ngvck_043	0.859624	ngvck_083	0.829168	ngvck_123	0.800707	ngvck_163	0.817234
ngvck_044	0.79271	ngvck_084	0.872405	ngvck_124	0.82741	ngvck_164	0.821403
ngvck_045	0.826229	ngvck_085	0.760503	ngvck_125	0.746659	ngvck_165	0.710488
ngvck_046	0.836969	ngvck_086	0.85487	ngvck_126	0.942152	ngvck_166	0.950248
ngvck_047	0.86469	ngvck_087	0.785574	ngvck_127	0.805325	ngvck_167	0.862672
ngvck_048	0.840169	ngvck_088	0.802513	ngvck_128	0.790965	ngvck_168	0.741967
ngvck_049	0.90374	ngvck_089	0.863611	ngvck_129	0.750409	ngvck_169	0.799065
ngvck_050	0.805312	ngvck_090	0.827641	ngvck_130	0.857436	ngvck_170	0.799495
ngvck_051	0.704289	ngvck_091	0.904243	ngvck_131	0.872555	ngvck_171	0.854514
ngvck_052	0.750496	ngvck_092	0.803578	ngvck_132	0.896051	ngvck_172	0.933774
ngvck_053	0.935332	ngvck_093	0.834778	ngvck_133	0.830718	ngvck_173	0.772713
ngvck_054	0.812124	ngvck_094	0.820635	ngvck_134	0.838589	ngvck_174	0.687992
ngvck_055	0.867598	ngvck_095	0.811333	ngvck_135	0.79889	ngvck_175	0.80426
ngvck_056	0.868336	ngvck_096	0.818489	ngvck_136	0.836009	ngvck_176	0.891901
ngvck_057	0.679785	ngvck_097	0.933964	ngvck_137	0.883729	ngvck_177	0.805876
ngvck_058	0.855351	ngvck_098	0.897759	ngvck_138	0.883737	ngvck_178	**0.66864**
ngvck_059	0.841996	ngvck_099	0.864706	ngvck_139	0.800752	ngvck_179	0.817758
ngvck_060	0.905574	ngvck_100	0.772584	ngvck_140	0.974362	ngvck_180	0.853839
ngvck_061	0.855445	ngvck_101	0.836188	ngvck_141	0.840758	ngvck_181	0.870896
ngvck_062	0.843742	ngvck_102	0.824486	ngvck_142	0.871869	ngvck_182	0.789751
ngvck_063	0.907973	ngvck_103	0.813456	ngvck_143	0.855663	ngvck_183	0.904679
ngvck_064	0.888358	ngvck_104	0.898684	ngvck_144	0.826324	ngvck_184	0.815941
ngvck_065	0.839282	ngvck_105	0.909746	ngvck_145	0.889468	ngvck_185	0.955751
ngvck_066	0.691726	ngvck_106	0.819268	ngvck_146	0.898341	ngvck_186	0.875103
ngvck_067	0.793397	ngvck_107	0.798243	ngvck_147	1.04783	ngvck_187	0.767535
ngvck_068	0.87561	ngvck_108	0.889952	ngvck_148	0.937647	ngvck_188	0.832419
ngvck_069	0.855727	ngvck_109	0.883636	ngvck_149	0.916563	ngvck_189	0.834388
ngvck_070	0.836216	ngvck_110	0.890187	ngvck_150	0.862537	ngvck_190	0.736419
ngvck_071	0.836522	ngvck_111	0.841355	ngvck_151	0.886537	ngvck_191	0.784291
ngvck_072	0.834869	ngvck_112	0.822144	ngvck_152	1.04194	ngvck_192	0.876888
ngvck_073	0.713289	ngvck_113	0.796133	ngvck_153	0.943327	ngvck_193	0.922199
ngvck_074	0.920724	ngvck_114	0.882668	ngvck_154	0.966463	ngvck_194	0.829006
ngvck_075	0.873066	ngvck_115	0.820483	ngvck_155	0.976824	ngvck_195	0.798529
ngvck_076	0.863213	ngvck_116	0.80588	ngvck_156	0.865403	ngvck_196	0.804606
ngvck_077	0.87679	ngvck_117	0.837689	ngvck_157	1.031274	ngvck_197	0.88721
ngvck_078	0.946965	ngvck_118	0.853458	ngvck_158	0.845869	ngvck_198	0.782963
ngvck_079	0.833724	ngvck_119	0.839827	ngvck_159	0.900087	ngvck_199	0.813735
ngvck_080	0.821638	ngvck_120	0.794348	ngvck_160	0.849555	ngvck_200	0.882488

Figure C-8: Graphical representation of Virus and Non-Virus Scores using ngvck_pp_group20_08 model

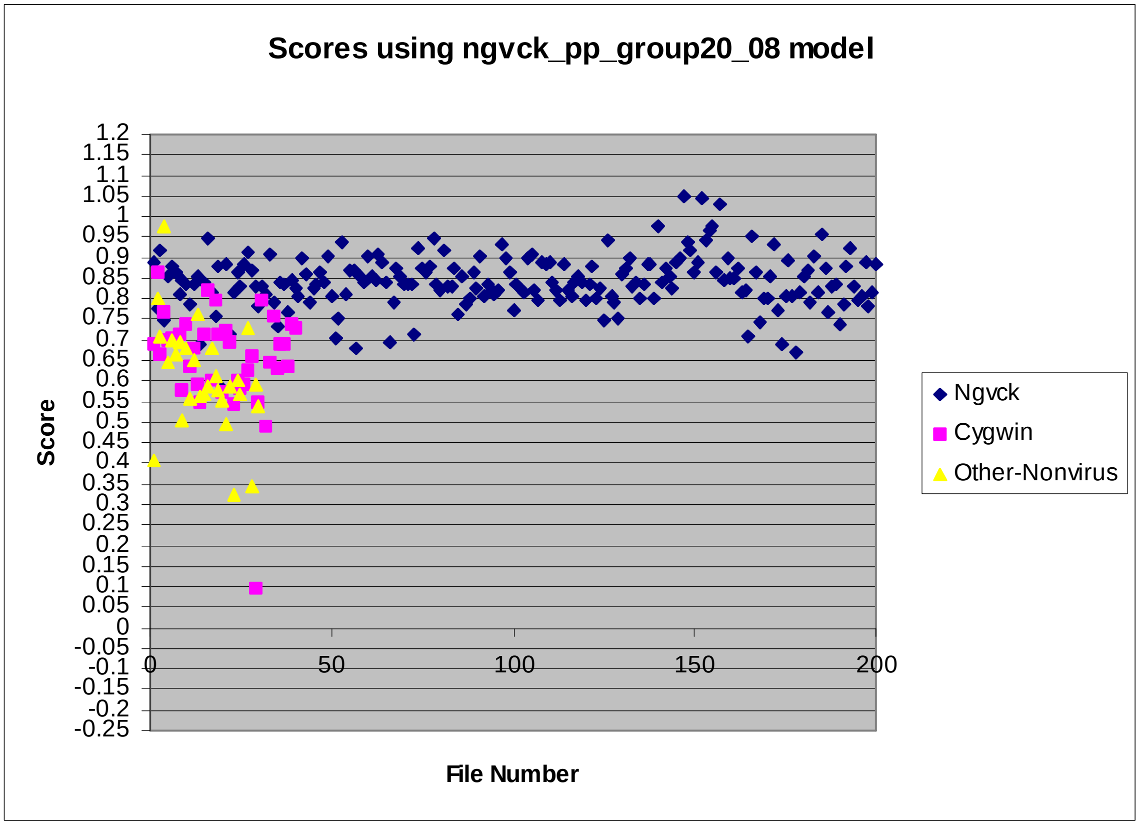


Table C-9.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_09 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.8467	cygwin_01	0.642955	nonvirus_01	0.395202
ngvck_002	0.862735	cygwin_02	**0.761702**	nonvirus_02	**0.713194**
ngvck_003	0.891343	cygwin_03	0.641485	nonvirus_03	**0.707139**
ngvck_004	0.8426	cygwin_04	**0.703916**	nonvirus_04	**0.816648**
ngvck_005	0.836165	cygwin_05	0.648203	nonvirus_05	0.608953
ngvck_006	0.804085	cygwin_06	0.650045	nonvirus_06	0.64707
ngvck_007	0.79117	cygwin_07	0.640925	nonvirus_07	0.605903
ngvck_008	0.767192	cygwin_08	0.665772	nonvirus_08	0.638037
ngvck_009	0.855232	cygwin_09	0.54619	nonvirus_09	0.51479
ngvck_010	0.920348	cygwin_10	0.670382	nonvirus_10	0.625015
ngvck_011	0.754918	cygwin_11	0.601528	nonvirus_11	0.520071
ngvck_012	0.873677	cygwin_12	0.64663	nonvirus_12	0.601335
ngvck_013	0.799154	cygwin_13	0.5737	nonvirus_13	**0.712667**
ngvck_014	0.760269	cygwin_14	0.531057	nonvirus_14	0.559963
ngvck_015	0.896827	cygwin_15	0.656245	nonvirus_15	0.564387
ngvck_016	0.883655	cygwin_16	**0.741775**	nonvirus_16	0.555113
ngvck_017	0.766549	cygwin_17	0.583765	nonvirus_17	0.61697
ngvck_018	0.735215	cygwin_18	**0.715955**	nonvirus_18	0.579912
ngvck_019	0.838088	cygwin_19	0.662721	nonvirus_19	0.552176
ngvck_020	**0.581149**	cygwin_20	0.545342	nonvirus_20	0.555523
ngvck_021	0.855421	cygwin_21	0.654253	nonvirus_21	0.494318
ngvck_022	0.797966	cygwin_22	0.592079	nonvirus_22	0.578756
ngvck_023	0.800017	cygwin_23	0.557306	nonvirus_23	0.420753
ngvck_024	0.771175	cygwin_24	0.561559	nonvirus_24	0.566911
ngvck_025	0.771735	cygwin_25	0.466957	nonvirus_25	0.53763
ngvck_026	0.824103	cygwin_26	0.545931	nonvirus_26	-2.766437
ngvck_027	0.837449	cygwin_27	0.583207	nonvirus_27	0.665153
ngvck_028	0.833139	cygwin_28	0.626975	nonvirus_28	0.366792
ngvck_029	0.795813	cygwin_29	0.129535	nonvirus_29	0.544359
ngvck_030	0.752853	cygwin_30	0.523881	nonvirus_30	0.494246
ngvck_031	0.887688	cygwin_31	**0.710977**		
ngvck_032	0.720614	cygwin_32	0.465987		
ngvck_033	0.896997	cygwin_33	0.590736		
ngvck_034	0.847262	cygwin_34	0.665903		
ngvck_035	0.727158	cygwin_35	0.608972		
ngvck_036	0.867133	cygwin_36	0.610567		
ngvck_037	0.911228	cygwin_37	0.603839		
ngvck_038	0.752406	cygwin_38	0.57922		
ngvck_039	0.8374	cygwin_39	0.669615		
ngvck_040	0.877902	cygwin_40	0.649991		

Table C-9.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_09 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.753302	ngvck_081	0.867793	ngvck_121	0.783076	ngvck_161	1.019969
ngvck_042	0.865901	ngvck_082	0.818889	ngvck_122	0.828114	ngvck_162	0.874079
ngvck_043	0.810561	ngvck_083	0.774827	ngvck_123	0.862592	ngvck_163	0.989519
ngvck_044	0.844247	ngvck_084	0.839646	ngvck_124	0.875405	ngvck_164	1.026387
ngvck_045	0.889764	ngvck_085	0.705039	ngvck_125	0.737176	ngvck_165	0.769648
ngvck_046	0.805807	ngvck_086	0.92345	ngvck_126	0.892286	ngvck_166	0.930501
ngvck_047	0.963534	ngvck_087	0.839838	ngvck_127	0.731885	ngvck_167	0.868818
ngvck_048	0.88873	ngvck_088	0.867872	ngvck_128	0.752611	ngvck_168	0.897586
ngvck_049	0.873463	ngvck_089	0.763462	ngvck_129	0.759153	ngvck_169	0.966084
ngvck_050	0.759452	ngvck_090	0.89777	ngvck_130	0.898989	ngvck_170	0.821347
ngvck_051	0.776751	ngvck_091	0.855034	ngvck_131	0.858691	ngvck_171	0.875476
ngvck_052	0.844896	ngvck_092	0.771678	ngvck_132	0.928073	ngvck_172	0.901643
ngvck_053	0.903033	ngvck_093	0.801366	ngvck_133	0.841615	ngvck_173	0.907257
ngvck_054	0.897038	ngvck_094	0.762874	ngvck_134	0.892001	ngvck_174	**0.69839**
ngvck_055	0.907983	ngvck_095	0.776704	ngvck_135	0.754002	ngvck_175	0.937481
ngvck_056	0.915907	ngvck_096	0.868156	ngvck_136	0.892931	ngvck_176	0.967634
ngvck_057	0.755941	ngvck_097	0.876522	ngvck_137	0.848813	ngvck_177	0.971157
ngvck_058	0.846208	ngvck_098	0.871824	ngvck_138	0.822338	ngvck_178	0.774363
ngvck_059	0.915899	ngvck_099	0.813551	ngvck_139	0.783055	ngvck_179	0.834452
ngvck_060	0.876291	ngvck_100	0.846303	ngvck_140	0.837417	ngvck_180	0.954403
ngvck_061	0.830285	ngvck_101	0.874378	ngvck_141	0.796149	ngvck_181	0.826118
ngvck_062	0.823961	ngvck_102	0.806742	ngvck_142	0.848936	ngvck_182	0.86639
ngvck_063	0.85802	ngvck_103	0.812342	ngvck_143	0.793869	ngvck_183	0.878642
ngvck_064	0.849756	ngvck_104	0.832634	ngvck_144	0.824358	ngvck_184	0.787076
ngvck_065	0.872632	ngvck_105	0.840578	ngvck_145	0.806727	ngvck_185	0.883247
ngvck_066	0.768171	ngvck_106	0.748349	ngvck_146	0.826007	ngvck_186	0.837598
ngvck_067	0.871324	ngvck_107	0.80111	ngvck_147	0.854007	ngvck_187	0.859214
ngvck_068	0.864064	ngvck_108	0.875219	ngvck_148	0.854706	ngvck_188	0.88798
ngvck_069	0.810433	ngvck_109	0.916041	ngvck_149	0.828465	ngvck_189	0.879918
ngvck_070	0.796965	ngvck_110	0.921398	ngvck_150	0.742959	ngvck_190	0.806717
ngvck_071	0.792842	ngvck_111	0.806872	ngvck_151	0.789481	ngvck_191	0.787896
ngvck_072	0.74882	ngvck_112	0.901229	ngvck_152	0.828123	ngvck_192	0.837309
ngvck_073	0.699248	ngvck_113	0.767215	ngvck_153	0.845602	ngvck_193	0.865728
ngvck_074	0.835851	ngvck_114	0.828212	ngvck_154	0.837855	ngvck_194	0.913318
ngvck_075	0.807648	ngvck_115	0.799522	ngvck_155	0.967379	ngvck_195	0.8774
ngvck_076	0.77548	ngvck_116	0.788343	ngvck_156	0.748469	ngvck_196	0.736768
ngvck_077	0.82793	ngvck_117	0.880958	ngvck_157	0.882318	ngvck_197	0.819572
ngvck_078	0.891998	ngvck_118	0.7795	ngvck_158	0.748594	ngvck_198	0.773445
ngvck_079	0.907019	ngvck_119	0.891867	ngvck_159	0.827073	ngvck_199	0.876383
ngvck_080	0.814825	ngvck_120	0.753408	ngvck_160	0.938722	ngvck_200	0.935838

Figure C-9: Graphical representation of Virus and Non-Virus Scores using ngvck_pp_group20_09 model

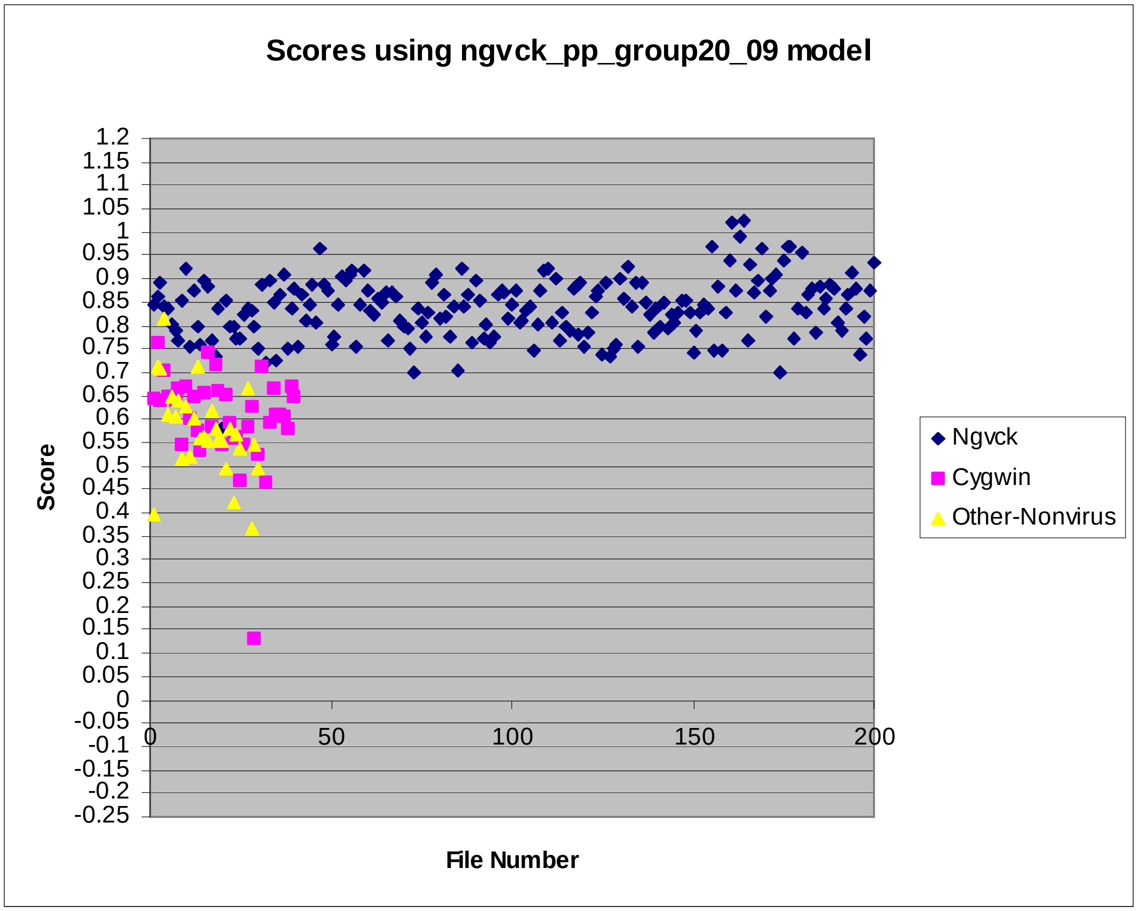


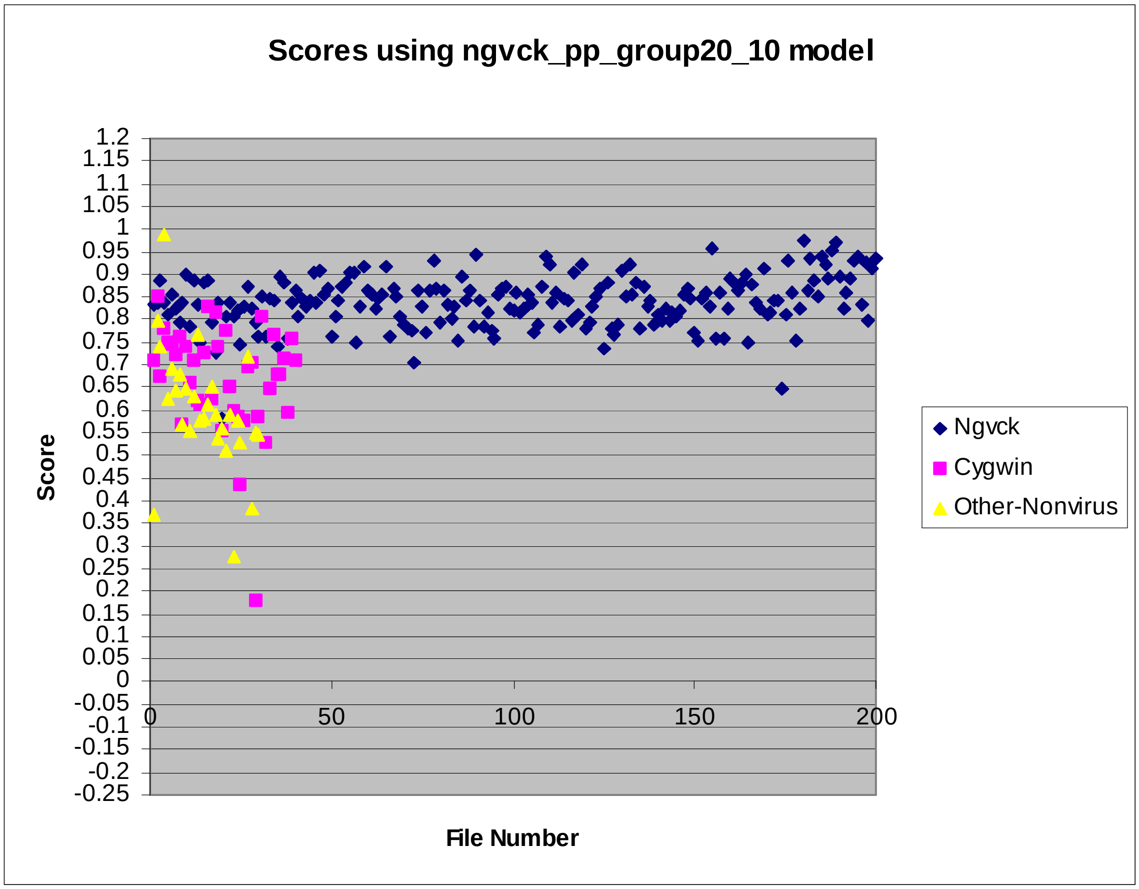
Table C-10.1 Scores of preprocessed Virus and Non Virus files using ngvck_pp_group20_10 model

NGVCK Virus variants after Pre-Processing		Non Virus files after Pre-Processing			
		Cygwin		Other Non Viruses	
File	Score	File	Score	File	Score
ngvck_001	0.835274	cygwin_01	**0.709146**	nonvirus_01	0.368329
ngvck_002	0.839564	cygwin_02	**0.851151**	nonvirus_02	**0.799011**
ngvck_003	0.884455	cygwin_03	0.675595	nonvirus_03	**0.74036**
ngvck_004	0.836423	cygwin_04	**0.778349**	nonvirus_04	**0.98869**
ngvck_005	0.812151	cygwin_05	**0.751269**	nonvirus_05	0.625614
ngvck_006	0.854471	cygwin_06	**0.745161**	nonvirus_06	0.690337
ngvck_007	0.823538	cygwin_07	**0.723689**	nonvirus_07	0.644455
ngvck_008	0.7911	cygwin_08	**0.762253**	nonvirus_08	0.676615
ngvck_009	0.835688	cygwin_09	0.568476	nonvirus_09	0.569572
ngvck_010	0.900649	cygwin_10	**0.740279**	nonvirus_10	0.646661
ngvck_011	0.786403	cygwin_11	0.661184	nonvirus_11	0.554642
ngvck_012	0.883959	cygwin_12	**0.707771**	nonvirus_12	0.628051
ngvck_013	0.831828	cygwin_13	0.621298	nonvirus_13	**0.767003**
ngvck_014	0.750639	cygwin_14	0.611181	nonvirus_14	0.574758
ngvck_015	0.88218	cygwin_15	**0.728782**	nonvirus_15	0.580729
ngvck_016	0.887437	cygwin_16	**0.82976**	nonvirus_16	0.610242
ngvck_017	0.794006	cygwin_17	0.62471	nonvirus_17	0.6516
ngvck_018	0.728453	cygwin_18	**0.816968**	nonvirus_18	0.589566
ngvck_019	0.836684	cygwin_19	**0.740482**	nonvirus_19	0.535808
ngvck_020	**0.580427**	cygwin_20	0.554621	nonvirus_20	0.56025
ngvck_021	0.807855	cygwin_21	**0.774337**	nonvirus_21	0.508226
ngvck_022	0.839337	cygwin_22	0.65265	nonvirus_22	0.587912
ngvck_023	0.805779	cygwin_23	0.600755	nonvirus_23	0.276487
ngvck_024	0.821028	cygwin_24	0.585671	nonvirus_24	0.575307
ngvck_025	0.745632	cygwin_25	0.436518	nonvirus_25	0.529595
ngvck_026	0.830191	cygwin_26	0.576001	nonvirus_26	-2.496257
ngvck_027	0.871291	cygwin_27	0.696381	nonvirus_27	**0.718465**
ngvck_028	0.82244	cygwin_28	**0.70685**	nonvirus_28	0.381314
ngvck_029	0.791279	cygwin_29	0.176976	nonvirus_29	0.54802
ngvck_030	0.763494	cygwin_30	0.586221	nonvirus_30	0.543744
ngvck_031	0.849665	cygwin_31	**0.807715**		
ngvck_032	0.762064	cygwin_32	0.530242		
ngvck_033	0.845671	cygwin_33	0.648863		
ngvck_034	0.841663	cygwin_34	**0.767342**		
ngvck_035	0.738297	cygwin_35	0.676187		
ngvck_036	0.895112	cygwin_36	0.678254		
ngvck_037	0.88164	cygwin_37	**0.712501**		
ngvck_038	0.757309	cygwin_38	0.593486		
ngvck_039	0.836564	cygwin_39	**0.755802**		
ngvck_040	0.864728	cygwin_40	**0.708962**		

Table C-10.2 Scores of preprocessed Virus files ngvck_041 to ngvck_200 using ngvck_pp_group20_10 model

NGVCK Virus Variants after Pre-Processing (Contd)							
File	Score	File	Score	File	Score	File	Score
ngvck_041	0.806136	ngvck_081	0.864506	ngvck_121	0.791963	ngvck_161	0.880509
ngvck_042	0.847177	ngvck_082	0.830975	ngvck_122	0.829968	ngvck_162	0.865064
ngvck_043	0.829665	ngvck_083	0.803309	ngvck_123	0.84861	ngvck_163	0.881594
ngvck_044	0.841277	ngvck_084	0.82883	ngvck_124	0.867851	ngvck_164	0.897337
ngvck_045	0.905322	ngvck_085	0.751691	ngvck_125	0.734984	ngvck_165	0.749164
ngvck_046	0.837817	ngvck_086	0.894088	ngvck_126	0.883431	ngvck_166	0.876397
ngvck_047	0.908585	ngvck_087	0.844116	ngvck_127	0.777839	ngvck_167	0.838498
ngvck_048	0.856655	ngvck_088	0.865981	ngvck_128	0.766265	ngvck_168	0.823362
ngvck_049	0.868916	ngvck_089	0.784684	ngvck_129	0.790593	ngvck_169	0.912443
ngvck_050	0.760482	ngvck_090	0.942007	ngvck_130	0.906221	ngvck_170	0.810481
ngvck_051	0.804519	ngvck_091	0.840213	ngvck_131	0.852502	ngvck_171	0.814819
ngvck_052	0.839949	ngvck_092	0.783553	ngvck_132	0.923051	ngvck_172	0.840636
ngvck_053	0.874903	ngvck_093	0.813714	ngvck_133	0.853154	ngvck_173	0.841052
ngvck_054	0.880146	ngvck_094	0.776534	ngvck_134	0.881862	ngvck_174	**0.64686**
ngvck_055	0.902571	ngvck_095	0.758418	ngvck_135	0.780832	ngvck_175	0.809198
ngvck_056	0.902833	ngvck_096	0.854568	ngvck_136	0.871903	ngvck_176	0.931621
ngvck_057	0.748321	ngvck_097	0.868806	ngvck_137	0.827791	ngvck_177	0.861389
ngvck_058	0.829101	ngvck_098	0.87281	ngvck_138	0.84194	ngvck_178	0.753529
ngvck_059	0.916347	ngvck_099	0.823875	ngvck_139	0.788465	ngvck_179	0.824825
ngvck_060	0.862244	ngvck_100	0.820958	ngvck_140	0.812093	ngvck_180	0.974242
ngvck_061	0.855147	ngvck_101	0.860169	ngvck_141	0.797802	ngvck_181	0.864314
ngvck_062	0.825158	ngvck_102	0.815833	ngvck_142	0.824642	ngvck_182	0.933638
ngvck_063	0.845969	ngvck_103	0.824356	ngvck_143	0.797001	ngvck_183	0.887501
ngvck_064	0.854689	ngvck_104	0.85324	ngvck_144	0.813329	ngvck_184	0.851609
ngvck_065	0.919003	ngvck_105	0.838511	ngvck_145	0.808395	ngvck_185	0.940524
ngvck_066	0.762422	ngvck_106	0.769408	ngvck_146	0.821483	ngvck_186	0.92356
ngvck_067	0.868012	ngvck_107	0.788327	ngvck_147	0.854361	ngvck_187	0.888566
ngvck_068	0.851694	ngvck_108	0.872985	ngvck_148	0.869809	ngvck_188	0.95117
ngvck_069	0.808513	ngvck_109	0.940077	ngvck_149	0.846805	ngvck_189	0.969765
ngvck_070	0.786868	ngvck_110	0.921029	ngvck_150	0.772844	ngvck_190	0.895132
ngvck_071	0.777821	ngvck_111	0.836868	ngvck_151	0.755694	ngvck_191	0.825318
ngvck_072	0.775516	ngvck_112	0.860339	ngvck_152	0.845488	ngvck_192	0.860018
ngvck_073	0.705139	ngvck_113	0.784173	ngvck_153	0.861015	ngvck_193	0.892635
ngvck_074	0.864175	ngvck_114	0.846718	ngvck_154	0.830788	ngvck_194	0.929118
ngvck_075	0.828592	ngvck_115	0.841961	ngvck_155	0.957069	ngvck_195	0.938228
ngvck_076	0.770082	ngvck_116	0.796383	ngvck_156	0.758893	ngvck_196	0.833595
ngvck_077	0.86613	ngvck_117	0.903696	ngvck_157	0.85778	ngvck_197	0.924956
ngvck_078	0.931194	ngvck_118	0.810071	ngvck_158	0.760051	ngvck_198	0.799816
ngvck_079	0.867157	ngvck_119	0.920812	ngvck_159	0.824571	ngvck_199	0.914229
ngvck_080	0.793652	ngvck_120	0.781737	ngvck_160	0.889094	ngvck_200	0.934642

Figure C-10: Graphical representation of Virus and Non-Virus Scores using ngvck_pp_group20_10 model


Scores using ngvck_pp_group20_10 model
Score
File Number
◆ Ngvck
■ Cygwin
▲ Other-Nonvirus