Faculty of Computer Science

ALEXANDRU IOAN CUZA UNIVERSITY of IAŞI

Alexandru Ioan Cuza University of Iaşi, Romania
Faculty of Computer Science

Thesis overview

Classification Algorithms for Malware Detection

Author:
Mihai Cimpoeşu

Supervisor:
Professor Dr. Henri Luchian

Table of Contents

1 Thesis Overview **3**

1.1 Contributions and outline of the thesis 3

1.2 Dronezilla. Malware behavior extraction. 5

1.3 Malware classification 9

1.4 Proactivity study on malware detection 18

1.5 Improving the PVM algorithm 19

1.6 PVM as a Single LP System 25

Bibliography **29**

Published articles

This thesis is submitted for being defended in fulfilment of the requirements for the Ph.D. degree from the Faculty of Computer Science, Alexandru Ioan Cuza University.

The complete list of journal and conference published papers that contribute to the content of this thesis can be found below:

- D. Gavriluţ, **M. Cimpoeşu**, A. Popescu, L. Ciortuz, *Malware detection using machine learning*, IMCSIT 2009, pages 735–741

- D. Gavrilut, **M. Cimpoeşu**, A. Popescu, L. Ciortuz, *Malware detection using machine learning*, Mathematica Balkanica Journal, Vol. 23, Fasc. 3–4, pages 209–229

- D. Gavrilut, **M. Cimpoeşu**, D. Anton, L. Ciortuz, *Malware Detection Using Perceptrons and Support Vector Machines* , Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World

- **M. Cimpoeşu**, D. Gavrilut, A. Popescu *The proactivity of Perceptron derived algorithms in malware detection*, Journal in Computer Virology, pages 133–140, 2012

- **M. Cimpoeşu**, C. Popa, *Dronezilla: designing an accurate malware behavior retrieval system*, Journal in Computer Virology, Vol. 8, No. 3, pages 109–116, 2012

- **M. Cimpoeşu**, C. Popa, *Dronezilla Automated behavioral analysis and testing framework*, EICAR 2012, Presses Techniques de l'ESIEA, ISBN 978-2-919106-00-4, pages 59–70
- A. Sucilă, **M. Cimpoeşu**, *A Distributed Solver for Dense Linear Feasibility Systems*, SYNASC 2012, pages 311–318
- **M. Cimpoeşu**, A. Sucilă, H. Luchian *A Statistical Binary Classier. Probabilistic Vector Machine*, Progress in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence, EPIA 2013, Angra do Herosmo, Azores, Portugal, September 9-12, 2013. Proceedings
- **M. Cimpoeşu**, A. Sucilă, H. Luchian *Probabilistic Vector Machine. Scalability through Clustering*, Progress in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence, EPIA 2013, Angra do Herosmo, Azores, Portugal, September 9-12, 2013. Proceedings
- **M. Cimpoeşu**, A. Sucilă, H. Luchian *Resolution of the Probabilistic Vector Machine Problem via a Single Linear Program*, 10th Workshop on Natural Computing and Applications, SYNASC 2013, Proceedings

1 | Thesis Overview

1.1 Contributions and outline of the thesis

The thesis brings contributions in two fields, *automated malware detection* and *classification algorithms*. The main target of my research was to build from the grounds-up a state of the art automatic malware detection framework based on machine learning algorithms. The hard constraint of having a zero false-positive and the performance required by a system to be used in practical, real-life scenarios made this project very challenging. After obtaining good results using perceptron-derived and SVM-related algorithms I concentrated my research on improving the performance of a newly introduced algorithm called Probabilistic Vector Machine. While trying to bring PVM into the competition with the best hyperplane classifiers I developed a distributed solver for the feasibility system in the initial formulation and then I simplified the entire model by reducing it to a single LP system.

The main contributions are summarized as follows:

- Chapter 1 of the thesis is an introductory chapter. It starts with a short survey of the current research context and continues with an introduction in into the malware detection research field. It defines the major malware categories such as: *worm*, *virus*, *Trojan horse*, *Rootkit*, *Bot*, *Spyware*; the major vectors of infection: *Exploitation of vulnerabilities in server software*, *Drive-by downloads*, *Social engineering*; following is a short overview of the *process of malware analysis*.

The second part of the chapter is dedicated to familiarize the reader with some important *supervised classification algorithms*: *Decision trees*, *The Perceptron*, *Artificial neural networks* and *Support Vector Machines.*

- The design and implementation of *Dronezilla*, an automatic malware behavior extraction environment implemented with real-hardware. This system was the corner-stone in the development of features used for detecting malware using Machine Learning. Details in Chapter 2.

- The development of both static and behavioral malware-related features for automated malware detection. Details in Chapter 3, Section 3.1.

- The design of the perceptron-derived algorithms used towards achieving a very low false positive rate when classifying malware. Details in Chapter3, Section 3.3.

- The design of experiments using classification algorithms for malware detection; the methodology and results of using a cascading ensemble of one-sided perceptrons and then one-sided SVMs are detailed in Chapter 3, Section 3.4 and Section 3.5.

- Experiments and practical optimizations for scaling-up purposes in the malware detection framework. Details in Chapter 3, Section 3.6.

- Designing a proactivity study on the resistence of automated detection models against malware evolution over a period of one year. Methodology and results in Chapter 4.

- For solving the system introduced by the original formulation of the PVM algorithm, an adaptable, general purpose distributed feasibility solver is introduced. Details in Chapter 5, Section 5.2.

- The implementation of a distributed feasibility solver for PVM based on the Twister map-reduce framework. Details on choosing algorithm parameters and results are presented in Chapter 5, Sections 5.2.4–5.2.9.

- A simplified model for the PVM algorithm is introduced. Reducing the statistical underlying model to a single LP system, allows PVM to compete with state of the art hyperplane classifiers. Details on the new model in Chapter 5, 5.3.

- Practical proof that PVM with the new statistical model is a competing hyperplane classification algorithm is given by providing comparrisons with state of the art algorithms on well known and artificial datasets. Details in Chapter 5., 5.3.5–5.3.6.

1.2 Dronezilla. Malware behavior extraction.

Among other type of features, in our malware detection system, we needed to record the dynamic behavior of malware. This means, that every sample needed to be executed to manifest itself as it was intended and the system should record it's behavior as features. For this type of task, historically, in the anti-malware industry, virtual machines were employed. Because most modern malware families have built-in mechanisms to establish that they are being run on a virtual machine, in such an environment, they manifest different behavior, such as: immediatelly terminating, pausing for a very long time, displaying a message-box signaling that the virtual machine was detected. Knowing this, we needed to create a system that would automatically execute and record real malware behavior.

Having automation and reliability set as our primary goals, we developed a framework environment based on real hardware. Within this environment one is allowed to automate most of the malware analysis tools which require accurate behavior of malware samples, which cannot be obtained using operating systems in virtual machines.

Among some of the most difficult constraints we faced while building this system was the speed of reverting from the infected operating system to clean snapshots or even to a brand new operating system. We overcame this step by booting the client/slave machines over network from a repository server that managed the hard-drive allocation. Moreover, the cloning, snapshotting and destroying hard disk images logic was created on top of the ZFS File System, which ran as a Free BSD kernel module. This also gave us a negligible delay time from shutting down one operating system to booting from a new hard-drive. The system also had to be scalable, secure and easy to attend. We discuss some of the interesting challenges we confronted with in achieving these tasks such as: scripting language controlled Power Distribution Units, video monitoring of client machines over network or private networking between each drone and its managing server. Throughout the rest of the thesis, the term *drone* will define a real hardware machine that is used to execute jobs scheduled by a managing server.

We present below step by step our progress in developing this framework including the choice of existing technologies, the needed changes and usage scenarios that range from modifying network interface card firmware, redesigning the AoE transmission protocol and drivers for every supported client operating system, to designing a web application for user interaction.

As the number of malware samples is vertiginously increasing by each day, the need for automated testing and information extracting environments for malware analysis is tremendous. There is a need to construct highly automated tools that will only require the human factor's attention for a very small number of samples, those that actually raise problems to the anti-malware solution in question. One of the existing implementation alternatives is represented by the use of existing virtual machines, offering a very similar environment to the real world hardware, although this exhibits more problems than it solves. The most critical problem is the virtual machine detection mechanisms that are built inside most of the modern malware packages. Besides the main intended purpose of this system –extracting malware behavior features– a side-effect of it's construction was the possibility to use it for general application performance testing. For a system constructed using this technology to be used for application performance testing it needs to have a *linear speed penalty* compared to the real hardware. As an example in this direction, consider more virtual machines (VM) running on the same hardware. If one VM is consuming more CPU, the rest of them could hang for a while. This leads to test cases that are hard to reproduce.

All of these limitations motivated us to construct a system and a framework whose purposes are:

- Real hardware usage. No virtual machines

- 100% automation of job running
- Limited human factor intervention
- Having a throughput similar to virtual machine technology;
- Increased speed in switching between an infected operating system and a clean one, a speed comparable with a virtual machine revert-to-snapshot

With these specifications in mind we constructed what we called *Dronezilla*, a master-slave system, based on existing technologies from the UNIX world, capable of being an environment which can automate the behavior analysis of malware samples and the quality assurance of software products. From our knowledge the system is a novelty in the field, raising the bar for the current techniques.

The most important technologies used in implementation:

- *ZFS* File System [1] – running as a FreeBSD kernel module – who made possible the creation of a remote repository with OS images, offering us very fast operations like cloning and snapshotting.
- The *AoE* and *iSCSI* network boot protocols are used here for loading operating system images from the remote repository on the machines attached to the system. The machines are called *drones* because they lack hard-drives.
- The video monitoring technologies, in our case being *DKVM* and *VNC*, through which we can monitor the client machines.

Some of the technologies used throughout building this system needed various modifications, without which the realization of

Dronezilla, as we planned it to be, was impossible. Here we include the modification of *gPXE* boot-loader firmware that we flashed into the network cards, the modification of ISC *DHCPD* Unix daemon or the modifications brought to the Windows AoE driver.

Dronezilla's core framework is formed of a series of daemons and scripts which are running the background logistics of the repository servers, plus a web application designed for managing and using the server. Python was the chosen programming language due to its productivity gains and lower development time.

1.3 Malware classification

Our *aim* is to overcome some of the big problems that the antivirus technologies face nowadays. Ultimately, these *problems* are expressed in terms of generic detection of malware, while getting as few false positives as possible.

In this chapter, we present a *framework* for malware detection aiming to get as few false positives as possible, by using a simple and a multi-stage combination (cascade) of different versions of the perceptron algorithm [8].

Furthermore results with a feature mapped perceptron and a kernelized perceptron are presented. Ultimatelly, a cascade one-sided Support Vector Machines (SVMs), combined with feature selection based on the F1 and F2 scores, is trained on a medium-size dataset consisting of clean and malware files. Cross-validation is then performed in order to choose the right values for parameters. Finally, tests are performed on another, non-related dataset. The obtained results were very encouraging.

1.3.1 Features for Machine Learning

For every binary file in the training and test datasets, a set of features/attributes was computed, based on many possible ways of analyzing a malware, for instance:

- *Behaviour characteristics in protected environments.* From the *Dronezilla* environment, we observed and monitored behavior characteristics of malware. These characteristics were afterwards introduced in a proprietary runtime emulation engine that is able to run the malware sample inside a protected environment and extract the features for every such sample in a matter of miliseconds.

- *File characteristics from the PE format point of view.* Malicious binary executables are regularly *modified* in order to make them smaller, packed and obfuscated. We monitor different techniques such as: abnormal sizes of each binary code section, which is very important; the number of of sections; the compiler that was used to generate the binary code; the presence of TLS data(code that is executed before the main thread is started); the number of DLL's references for dynamic linking; the presence of certain resource information embedded inside the PE executable, etc.

- *File format from a geometrical point of view.* For a trained eye, actually seeing a new binary malware, it is very easy to say if something is suspicious. We noted down all the abnormalities that we observed when *looking* at a binary and used them as features for the learning framework.

- *File packer, obfuscator or protector type.* An important practical observation was that most of the malware code nowadays comes packed, obfuscated or protected(i.e. protected from manual step by step debugging). We transformed some of the static signatures detecting known packers/obfuscators/protectors into features contributing in the identification of malware.

- *Package installer type.*
- *File content information retrieved from imports, exports, resource directory or from different strings that reside in the data section of the file.*
- *Compiler specific features.*

The total number of file attributes that we defined –excluding those related to imported functions– was around 800, but for the scope of this thesis only 308 *boolean* attributes were used.

1.3.2 Datasets and algorithms

We used three datasets: a *training* dataset, a *test* dataset, and a *"scale-up"* dataset. The number of malware files and respectively clean files in these datasets is shown in the first two columns of Table 1.1. As stated above, our main goal is to achieve malware detection with only a few (if possible 0) false positives, therefore the clean files in this dataset (and also in the scale-up dataset) is much larger than the number of malware files.

The clean files in the training database are mainly system files (from different versions of operating systems) and executable and library files from different popular applications. We also use clean files that are packed or have the same form or the same geometrical similarities with malware files (e.g use the same packer) in order to better train and test the system.

Table 1.1: Number of files and Unique Combinations of Feature Values in the Training, Test, and Scale-Up Datasets.

	Files		Unique combinations	
Database	malware	clean	malware	clean
Training	27475	273133	7822	415
Test	11605	6522	506	130
Scale-up	approx. 3M	approx. 180M	12817	16437

The malware files in the training dataset have been taken from the Virus Heaven collection(henceforth denoted VH). The test dataset contains malware files from the WildList (henceforth denoted WL) collection and clean files from different operating systems (other files that the ones used in the first database).

The modified Perceptron algorithms that we used are fully described in the thesis Section 3.3. We present here only the important training subroutine of *One-Sided Perceptron* 1 algorithm that helped with lowering the False Positive rate.

Algorithm 1 One-Sided Perceptron

```
$NumberOfIterations \leftarrow 0$
$MaxIterations \leftarrow 100$
**repeat**
  **Train** ($R$, 1, -1)
  **while FP**($R$) $> 0$ **do**
    **Train** ($R$, 0, -1)
  **end while**
  $NumberOfIterations \leftarrow NumberOfIterations + 1$
**until** (**TP**($R$) $= NumberOfMalwareFiles$) or
    ($NumberOfIterations = MaxIterations$)
```

For what we call the *mapped one-sided perceptron*, we will use the previous perceptron algorithm, except we first map all our features in a different space using a simple feature generation algorithm, presented in the thesis as Algorithm 3.

Finally, we used the same one-sided perceptron (Algorithm 1), but in the dual form [3] and with the training entry mapped into a larger feature space via a kernel function K [9]. The resulting *kernelized one-sided perceptron* is the Algorithm 2 given below.

Algorithm 2 Kernelized One-Sided Perceptron

```
for $i = 1$ to $n$ do
    $\Delta_i \leftarrow 0$
    $\alpha_i \leftarrow 0$
end for
for $i = 1$ to $n$ do
    if $(label_i \times \sum_{j=1}^{n}(\alpha_j \times K(i,j))) \leq 0$ then
        $\Delta_i \leftarrow \Delta_i + label_i$
    end if
end for
for $i = 1$ to $n$ do
    $\alpha_i \leftarrow \alpha_i + \Delta_i$
    $\Delta_i \leftarrow 0$
end for
```

1.3.3 Results with Perceptron algorithms

Cross-validation tests for 3, 5, 7, and 10 folds were performed for each algorithm (COS-P, COS-P-Map, COS-P-Poly and COS-P-Radial) on the *training* dataset. For each algorithm, we used the best result from maximum 100 iterations.

The cross-validation results found in Table 1.2 show that although the COS-P-Poly4 algorithm has the best malware detection rate (i.e sensitivity) on training dataset, the number of false alarms produced by this algorithm is much higher than the one

Table 1.2: 5-fold Cross-validation Results on the Training Dataset.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
COS-P	1342	**5**	85.83%	**93.98%**	86.24%
COS-P-Map-F1	1209	18	97.25%	74.09%	95.97%
COS-P-Map-F2	1212	17	96.98%	77.50%	95.83%
COS-P-Poly2	1518	23	97.05%	71.57%	95.76%
COS-P-Poly3	1532	29	97.95%	64.10%	**96.25%**
COS-P-Poly4	1533	31	**98.01%**	61.69%	96.18%
COS-P-Radial	1153	33	97.42%	63.37%	95.70%

obtained for the COS-P algorithm. (Note that the number of files that are actually detected is much higher since the algorithm works with unique combinations of features and not with actual files.)

Table 1.3: Results on the Test Dataset.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
COS-P	356	3	68.73%	**97.46%**	74.06%
COS-P-Map-F1	356	**2**	83.76%	96.97%	85.54%
COS-P-Map-F2	357	**2**	83.22%	97.14%	85.17%
COS-P-Poly2	455	9	87.84%	92.37%	88.68%
COS-P-Poly3	466	19	89.96%	83.90%	**88.84%**
COS-P-Poly4	465	20	**89.77%**	83.05%	88.52%
COS-P-Radial	264	19	89.13%	86.92%	88.68%

The results for the *test* dataset (Table 1.3) show that both COS-P-Map-F1 and COS-P-Map-F2 algorithms produce good results, with a good specificity (83%) and very few (2) false positives, even if the malware distribution in this dataset is different from the one in the training dataset.

From the technical point of view, the most convenient algorithms are the cascade one-sided perceptron (COS-P) and its explicitly mapped version (COS-P-Map).

1.3.4 Results with One-Sided SVMs

We first trained the classical SVM algorithm [2, 3] on the VH dataset (Tables 1.4 and 1.5). The denotations for the different versions of the SVM algorithm correspond to those introduced in the precedent subsection. The test results on the WL dataset, as shown in Table 1.7 provided a better detection rate compared to all Perceptron algorithms mentioned in the previous subsection. However, the number of false positives is much higher now. This why we opted for the one-sided version of the SVM classification algorithm, henceforth abbreviated OS-SVM. Technical details for running OS-SVM are given in Section 3.5 of the thesis.

Training results with OS-SVM are shown in Tables 1.6. The test results obtained by using this method were very encouraging regarding the false positive rate, but not the true positive rate, as it can be seen in Table 1.8.

We applied the cascading methodology for One-Sided SVM in conjunction with each kernel function presented in the previous subsection. The test results, given in Table 1.9, show an increased detection rate and very few false positives compared to both those obtained by the cascade one-sided perceptrons (Table 1.3) and the non-cascade one-sided SVMs (Tables 1.7 and 1.8).

The main goal was to build a machine learning framework that generically detects as much malware samples as possible, with the tough constraint of having a zero false positive rate. Using just Perceptron algorithms, we were very close to our goal, although

Table 1.4: 5-fold Cross-validation, SVM Results on the Training (VH) Set.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
SVM-Map-F1	6149	68	97.46%	74.81%	96.53%
SVM-Map-F2	6179	74	97.66%	76.58%	96.66%
SVM-Poly2	7728	94	97.64%	70.72%	96.59%
SVM-Poly3	7741	81	97.36%	71.58%	96.47%
SVM-Poly4	7744	78	96.99%	69.05%	96.14%
SVM-Radial	7747	75	**97.93%**	**76.92%**	**97.10%**

Table 1.5: 10-fold Cross-validation, SVM Results on the Training (VH) Set.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
SVM-Map-F1	6159	58	97.55%	78.11%	96.76%
SVM-Map-F2	6173	80	97.67%	75.23%	96.58%
SVM-Poly2	7729	93	97.54%	70.19%	96.50%
SVM-Poly3	7801	21	96.17%	**83.06%**	95.97%
SVM-Poly4	7813	9	95.41%	80.85%	95.33%
SVM-Radial	7750	72	**97.84%**	77.14%	**97.05%**

Table 1.6: 5-fold CV, OS-SVM Results on the Training (VH) Set.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
OS-SVM-Map-F1	743.4	1.6	**59.78%**	97.83%	**61.87%**
OS-SVM-Map-F2	657.4	1	52.56%	98.72%	55.27%
OS-SVM-Poly2	814.2	**0.6**	52.04%	**99.27%**	54.41%
OS-SVM-Poly3	790.2	1.4	50.5%	98.31%	52.9%
OS-SVM-Poly4	909.6	2.8	58.14%	96.62%	60.07%
OS-SVM-Radial	701.6	**0.6**	44.84%	**99.27%**	47.57%

Table 1.7: SVM Results on the Test (WL) Dataset.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
SVM-Map-F1	405	10	95.29%	84.62%	93.88%
SVM-Map-F2	401	10	93.47%	85.51%	92.37%
SVM-Poly2	490	23	96.84%	82.17%	93.86%
SVM-Poly3	503	35	99.41%	72.87%	94.02%
SVM-Poly4	506	89	**100.00%**	31.01%	85.98%
SVM-Radial	486	18	96.05%	**86.05%**	**94.02%**

Table 1.8: OS-SVM Results on the Test (WL) Dataset.

Algorithm	**TP**	**FP**	**SE**	**SP**	**ACC**
OS-SVM-Map-F1	302	**0**	71.06%	**100%**	74.9%
OS-SVM-Map-F2	311	**0**	**72.49%**	**100%**	**76.31%**
OS-SVM-Poly2	321	**0**	63.44%	**100%**	70.87%
OS-SVM-Poly3	324	**0**	64.03%	**100%**	71.34%
OS-SVM-Poly4	334	**0**	66.01%	**100%**	72.91%
OS-SVM-Radial	330	**0**	65.22%	**100%**	72.28%

Table 1.9: Results for Cascade One-Sided SVMs on the Test (WL) Dataset.

Algorithm	**ITER**	**TP**	**FP**	**SE**	**SP**	**ACC**
COS-SVM-Map-F1	181	283	0	66.58	**100**	71.02
COS-SVM-Map-F2	362	317	2	73.89	97.1	77.1
COS-SVM-Poly2	276	362	3	71.54	97.67	76.85
COS-SVM-Poly3	157	421	16	83.2	87.59	**84.09**
COS-SVM-Poly4	593	375	4	74.11	96.89	78.74
COS-SVM-Radial	625	407	13	**80.43**	89.92	82.36

we still have a non-zero false positive rate. In order for this framework to become part of a highly competitive commercial product, a number of deterministic exception mechanisms have to be added.

As of now, malware detection via machine learning will not replace the standard detection methods used by anti-virus vendors, but will come as an important addition to them. Any commercial anti-virus product is subject to certain speed and memory limitations, therefore the most reliable algorithms among those presented here are the cascade one-sided perceptron and and its explicitly mapped variant.

Since most anti-malware solutions manage to have a detection rate of over 90% according to AV-TEST [7], it follows that an increase of the total detection rate of $3\% - 4\%$ as the one produced by our algorithms, is very significant.

1.4 Proactivity study on malware detection

In this study we use the representation of data described above to train multiple machine-learning algorithms. We used a combination of different approaches in order to obtain results with different characteristics, such as: algorithms that exhibit a very big detection rate with the downside of having a high rate of false positives, algorithms that have very strict policy in restricting false positives but with a lower the detection –true positive– rate or methods that have both high true positive and true negative rate, but in order to train them on large scale databases, a huge amount of computational resources is needed. We have also used ensembles of multiple trained models such as voting or cascading schemes.

With this experiment we tried to give the community the answer to the following question: "What is the correlation between the size [1] of a database and the power of the extracted models to pro-actively detect malware for a limited period of time (14 weeks in this case)". We used a large amount of data and extracted a database of feature vectors for every file. To our knowledge, nobody else in the literature ever attempted to present results using a database of such size. We used our own implementations of the algorithms in order to sustain this quantity of data and studied the resistance of the trained models in time against the evolution of malware strains.

We have presented several machine learning methodologies for distinguishing between malware and clean files and used them to study the power of every algorithms against the ever-changing characteristics of malware in-the-wild. We have shown multiple methods with different characteristics. Some of them work on a shorter time-span giving good results, like for example the *Cascaded Ensemble* and we have shown methods that are resistant in time, like the *OneSide Perceptron*. All of the presented algorithms are suited for real life practical use as long as one can respect the characteristics of the method and exploit it's benefits. Our study showed that after approximately one month, the malware characteristics have visible changes and we recommend to replace the production models with newly trained ones.

1.5 Improving the PVM algorithm

This chapter outlines the progress done starting from the original PVM formulation by my coleague Andrei Sucila in [10]. The original PVM formulation is presented in Section 5.1 of the thesis.

[1] The size of the database here is an indicator of the diversity of the malware samples involved.

The main target was to scale-up the PVM algorithm in order to deal with larger quantities of input data. Section 5.2 of the thesis describes the efforts in optimizing the solver of the feasibility system needed for PVM training. A simpler reformulation of the PVM system allows for smaller traning times – to solve the PVM problem, a single linear programming system needs to be solved. All the steps that lead to this point are described in Section 5.3.

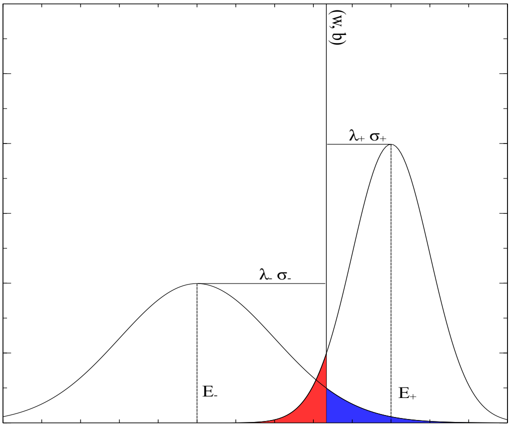

Figure 1-1: Induced distance distributions. The hyperplane is sought such as to minimize the maximum between the red and blue areas, which correspond to the FN and FP probabilities. Equivalently, the hyperplane has to maximize the minimum between λ_+ and λ_-.

The PVM optimization problem is equivalent to solving:

$$\begin{cases} minmax\{\frac{\sigma_+}{E_+}, \frac{\sigma_-}{E_-}\} \\ b + \frac{1}{|S_+|}\sum_{x_i \in S_+} < w, x_i > = E_+ \\ -b - \frac{1}{|S_-|}\sum_{x_i \in S_-} < w, x_i > = E_- \\ E_+ \geq 1, E_- \geq 1 \\ | < w, x_i > +b - E_+| \leq \sigma_+^i, \forall x_i \in S_+ \\ | < w, x_i > +b + E_-| \leq \sigma_-^i, \forall x_i \in S_- \\ \sigma_+ = \frac{1}{|S_+|-1}\sum_{x_i \in S_+} \sigma_+^i \\ \sigma_- = \frac{1}{|S_-|-1}\sum_{x_i \in S_-} \sigma_-^i \end{cases} \tag{1.1}$$

Note that, besides the objective function, system (1.1) uses only linear equations.

A few important properties of the model thus far are that it has a direct connection to the generalization error embedded in the objective function and that it is likely to be resillient to outliers, as it is based on a statistical model of the training data and, as such, has a natural mechanism of dealing with outliers.

Also note that, because of the way the system is built, it does not require for S_+ and S_- to be linearly separable, as would a hard margin SVM, and does not require special treatment for classification errors, thus avoiding the introduction of a tradeoff term in the objective function.

Replacing the scalar products in system (1.1) with a kernel function, we obtain the PVM model that uses kernel functions when separability is difficult in the original space.

$$\begin{cases} \min\max\{\frac{\sigma_+}{E_+}, \frac{\sigma_-}{E_-}\} \\ b + \sum_{i=1}^{m} [\alpha_i \cdot \frac{1}{|S_+|} \sum_{x_j \in S_+} K(x_i, x_j)] = E_+ \\ -b - \sum_{i=1}^{m} [\alpha_i \cdot \frac{1}{|S_-|} \sum_{x_j \in S_-} K(x_i, x_j)] = E_- \\ |\sum_{x_i \in S} \alpha_i K(x_i, x_j) + b - E_+| \le \sigma_+^j \ , \forall x_j \in S_+ \\ |\sum_{x_i \in S} \alpha_i K(x_i, x_j) + b + E_-| \le \sigma_-^j \ , \forall x_j \in S_- \\ \frac{1}{|S_+|-1} \sum_{x_i \in S_+} \sigma_+^i = \sigma_+ \\ \frac{1}{|S_-|-1} \sum_{x_i \in S_-} \sigma_-^i = \sigma_- \\ \sigma_+ \le t \cdot E_+ \\ \sigma_- \le t \cdot E_- \\ E_+ \ge 1, \ E_- \ge 1 \end{cases} \tag{1.2}$$

1.5.1 Distributed PVM solver

Because of the high computational demands of the initial solver for PVM, we introduce a distributed solver based on the Distaince Weighted Projection Operator. Details on DWPO are given in Section 5.3.2 of the thesis. The solver is implemented using the Twister [5] map-reduce [4] framework.

For machine learning algorithms that need multiple iterations over the same static data, the most important feature that Twister introduces is the ability to run multiple iterations on the same data. That is, once the mappers and the reducers are configured with pre-computed data, each computing unit keeps their portion of the matrix in memory at all times. Because there is no time dedicated for loading data at each iteration and all communication between nodes is done via the pub/sub messaging bus and not through files from a virtual file-system, the cluster resources are used at full capacity.

A high-level overview of the Twister programming model is

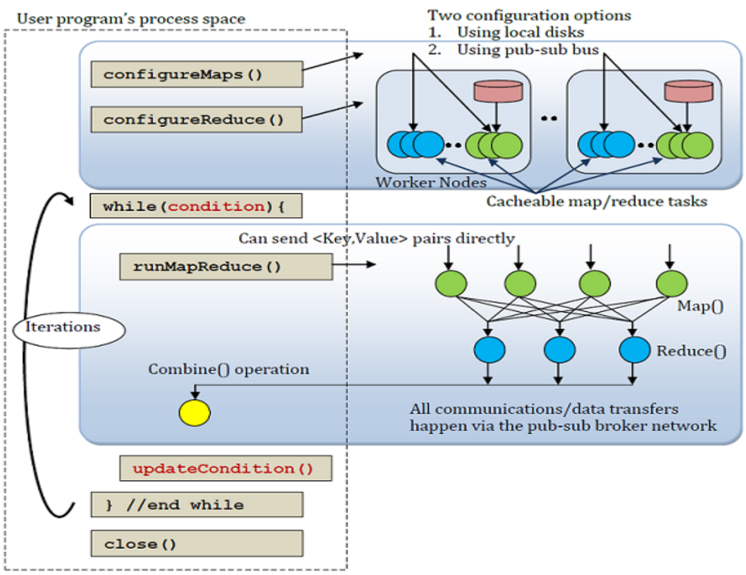

Figure 1-2: Twister high level programming model
[2]Source: http://www.iterativemapreduce.org

described in Figure 1-2. As it can be seen, Twister does not use a virtual file-system to load data from. The partitioning and copying of the static data is done via Unix Bash scripts. If this solver weren't such a specific task, the lack of the distributed virtual file-system, could be considered a flaw in a Big Data processing framework, but because we only load all the static data once and use it multiple times, this is not a problem here.

The steps taken when training a new PVM model are:

- Transform all nominal attributes in numeric ones
- Normalize dataset
- Copy traning dataset on all machines in the cluster
- Launch Twister job that for static data pre-processing. This task signals each computing node (the mappers) to load the dataset and pre-compute the slice of the Gram Matrix that corresponds to this node. The Gram Matrix here contains the distances from each data record to all others.
- Launch Twister job to train using the technique described in Section 5.2.2 of the thesis. Each mapper loads one block of data and in each iteration, when signaled, computes the update for the current block. The update from the mappers is transmitted to the reducers which just pass it further to the final Reduce stage, called Combiner. The Combiner runs on the main machine and is responsible for making the global update and send it to all mappers through the pub/sub bus. The mappers are signaled to start working again.

1.6 PVM as a Single LP System

The classifier described thus far is obtained as the result of solving a series of linear feasibility problems. This approach allowed preliminary validation of the concept, but when dealing with practical datasets, several problems became immediately apparent:

1. The final linear feasibility problems lead to ill–conditioned systems for which the feasible region consists of a single line in the solution space.

2. The precision can suffer badly. Especially in kernel problems, where small variations in the objective value (to the order of 10^{-6}) can lead to large variations in accuracy (to the order of 2%).

3. The resolution time can be large. Even with hot starts, the linear solvers can take very long to solve the linear feasibility problems.

While items 2 and 3 can be separately dealt with, kernel problems would still tend to take very long to solve. This is because a large number of linear feasibility problems would have to be resolved for the precision to be high enough to offset the dropoff in accuracy.

To resolve these problems, a better mathematical approach was required. The idea is to show that the problem has an equivalent linear fractional formulation and then convert the linear fractional program to a simple linear program. The steps needed to reach the final form of the Single LP PVM accompanied by proofs are detailed in Section 5.3 of the thesis. The final model that uses kernel functions has the following form:

$$\begin{cases} \min \frac{1}{|S_+|-1} \sum_{i \in S_+} \sigma_+^i + \frac{1}{|S_-|-1} \sum_{i \in S_-} \sigma_-^i \\ |\sum_{j=1}^m \alpha_j (K_+^j - K(x_i, x_j))| \leq \sigma_+^i, \forall x_i \in S_+ \\ |\sum_{j=1}^m \alpha_j (K_-^j - K(x_i, x_j))| \leq \sigma_-^i, \forall x_i \in S_- \\ \sum_{i=1}^m \alpha_i (K_+^i - K_-^i) = 1 \end{cases} \tag{1.3}$$

Note that the hyperplane offset no longer appears among the variables of this system. It can, instead, be found at the end by resolving the equation $\frac{\sigma_+}{E_+} = \frac{\sigma_-}{E_-}$.

Problem (1.3) is a linear program that can be solved with one of the many available solvers. It is important to note that it no longer leads by default to ill conditioned systems. Also, solving this will give a solution to the original PVM problem without any loss in accuracy.

1.6.1 Results on UCI ML Datasets

PVM has been tested on a series of data sets originating from the University of California–Irvine Machine Learning (UCI ML) database. The sets used for testing are the ones which have also been used in [6] and are decribed in Table 1.10.

Tables 1.11 and 1.12 show the results of comparing PVM with the LSTSVM, TSVM, GEPSVM and PSVM classifiers. For PVM the selection of the kernel and bias parameters was done using a grid search. Each combination of parameters was evaluated using 5 tenfold cross-validations. The final evaluation of the best parameters selected in this manner was done by running 100 tenfold cross-validations, equivalent to 1000 runs, and recording the average and standard deviation of these 1000 runs. The results for the other classifiers have been taken from [6] where the same tenfold testing procedure is used, albeit using a slightly lower

Dataset	Records	Features	Positive	Negative
Hepatitis	156	19	32	123
WPBC	199	34	47	151
Sonar	209	60	111	97
Heart-statlog	270	14	150	120
Heart-c	303	14	139	164
Bupa Liver	345	7	200	145
Ionosphere	351	34	126	225
Votes	434	16	267	167
Australian	690	14	307	383
Pima-Indian	768	8	500	268

Table 1.10: UCI ML datasets description

Dataset	PVM	LSTSVM	TSVM	GEPSVM	PSVM
Hepatitis	**87.151**	84.28	83.73	79.28	78.57
WPBC	78.853	81.66	**82.22**	80	80.55
Sonar	86.995	**90.52**	90	80	90
Heart-statlog	77.548	85.18	85.84	**86.52**	70.74
Heart-c	77.119	**83.79**	82.17	70.37	70.68
Bupa Liver	73.021	74.84	**75.15**	68.18	74.84
Ionosphere	92.903	**96.17**	**96.17**	84.41	95
Votes	**96.54**	96.19	95.95	94.5	95.95
Australian	**83.623**	76.17	75.8	69.55	73.97
Pima-Indian	**77.133**	75.33	75.74	75.33	76.8

Table 1.11: Comparison between PVM, LSTSVM, TSVM, GEPSVM, PSVM on the UCI ML datasets on the RBF kernel.

Dataset	PVM	LSTSVM	TSVM	GEPSVM	PSVM
Hepatitis	0.94	10.24	6.25	5.2	**0.24**
WPBC	**0.64**	6.95	6.82	5.97	3.92
Sonar	**1.37**	7.36	7.5	5.97	7.21
Heart-statlog	**1.84**	5.23	6.52	7.36	6.86
Heart-c	**1.21**	5.87	5.21	8.90	7.66
Bupa Liver	**0.92**	6.85	6.51	6.2	9.04
Ionosphere	**1.36**	3.68	3.9	6.2	4.17
Votes	**0.39**	2.79	3.37	3.37	2.25
Australian	**0.94**	5.36	4.91	5.37	6.16
Pima-Indian	**0.31**	4.67	5.2	4.91	3.83

Table 1.12: Comparison between PVM, LSTSVM, TSVM, GEPSVM, PSVM on the UCI ML datasets on the RBF kernel.

number of runs.

Table 1.11 shows the results using the RBF kernel. PVM has the most data sets on which it obtains the best results. It is important to note that PVM produces the best results for the largest datasets, where the statistical information gains more and more relevance. What is a bit surprising is that one of the smallest datasest, Hepatitis, also produces a win. Examining the runs on this dataset shows that most of the trainings would degenerate into zero average deviation, showing that this case is worth discussing.

Table 1.12 shows the standard deviations for the results of the classifiers. It is important to note that the standard deviations for PVM are usually much lower than the other classifiers, with the exception of Hepatitis. This indicates that, for different folds of a tenfold, the result of the training is very similar, resulting in roughly the same accuracy when testing using the developed model. It is an argument for the stability of the classifier.

Bibliography

[1] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The zettabyte file system. Technical report, Sun Microsystems, 2003. (Cited on page 8.)

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. (Cited on page 15.)

[3] Nello Cristianini and John Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, March 2000. (Cited on pages 13 and 15.)

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. (Cited on page 22.)

[5] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM. (Cited on page 22.)

[6] Arun M. Kumar and M. Gopal. Least squares twin support vector machines for pattern classification. *Expert Systems with Applications: An International Journal*, 36(4):7535–7543, 2009. (Cited on page 26.)

[7] Andreas Marx. Av-test anti-malware solutions testing. http://www.av-test.org/no_cache/en/tests/test-reports. [Online; accessed 15-June-2013]. (Cited on page 18.)

[8] F. Rosenblatt. The perception: a probabilistic model for information storage and organization in the brain. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: foundations of research*, pages 89–114. MIT Press, Cambridge, MA, USA, 1988. (Cited on page 9.)

[9] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002. (Cited on page 13.)

[10] Andrei Sucilă and Henri Luchian. Probabilistic vector machine. In *The 7th International Conference on Data Mining DMIN'11, Las Vegas, Nevada, USA*, pages 198–202. CSREA Press, 2011. (Cited on page 19.)