**UNITED STATES PATENT AND TRADEMARK OFFICE**

---

BEFORE THE PATENT TRIAL AND APPEAL BOARD

---

TREND MICRO, INC.

v.

WEBROOT, INC.

---

Petition for *Inter Partes* Review of

U.S. Patent No. 10,599,844

---

# DECLARATION OF INGRID HSIEH-YEE, PH.D.

## TABLE OF CONTENTS


I. Introduction ..... 3

A. Qualifications and Professional Experience ..... 3

B. Scope of This Declaration ..... 11

C. Person of Ordinary Skills in the Arts ..... 12

D. Evidence Considered in Forming My Opinions ..... 13

II. Authenticity and Public Availability of *Tahan* (EX1003) ..... 15

A. Authentication ..... 15

B. Internet Archive Records ..... 16

C. Usage Record ..... 19

D. Summary of My Opinion on *Tahan* ..... 20

III. Authenticity and Public Availability of *Menahem* (EX1004) ..... 20

A. Authentication ..... 20

B. Library of Congress Records ..... 23

C. Internet Archive Records ..... 27

D. Internet Archive Affidavit (EX1019) ..... 28

E. Citation Record ..... 29

F. Summary of My Opinion on *Menahem* ..... 29

IV. Conclusion ..... 30

## I. Introduction

I, Ingrid Hsieh-Yee, Ph.D., do hereby declare as follows:

1. I have personal knowledge of all of the matters about which I testify in this declaration.

2. I have been retained as an independent expert witness on behalf of Trend Micro, Inc. ("TREND") for proceedings before the United States Patent and Trademark Office ("PTO") regarding the authenticity and public availability of certain prior art reference in relation to US Patent No. 10,599,844 ("the '844 patent").

3. I am being compensated for my work in this matter at my customary hourly rate. I am also being reimbursed for any reasonable expenses associated with my work and testimony in this investigation. My compensation is not contingent on the results of my study, the substance of my opinions, or the outcome of this matter.

### A. Qualifications and Professional Experience

4. My complete qualifications and professional experience are described in my academic curriculum vitae (**Appendix A**). The following is a brief summary of my relevant qualifications and professional experience.

5. In 2022 I was conferred the title of Professor Emerita after 32 years as a professor in the Department of Library and Information Science of the Catholic University of America. I have experience working in an academic library, a medical

library, and a legislative library. I hold a Ph.D. in Library and Information Studies from the University of Wisconsin-Madison and a Masters in Library and Information Studies from the University of Wisconsin-Madison.

6. I am an expert on library cataloging and classification and have published two editions on this subject, *Organizing Audiovisual and Electronic Resources for Access*: *A Cataloging Guide* (2000 and 2006). I taught a variety of courses, including Cataloging and Classification, Advanced Cataloging and Classification, Organization of Internet Resources, Organization of Information, Digital Content Creation and Management, Metadata, Internet Searches and Web Design, Information Literacy Instruction, Advanced Information Retrieval and Analysis Strategies, and The Information Professions in Society. I am familiar with metadata schema design and implementation. In my teaching I covered the design and implementation of metadata in libraries, databases, search engines, digital repositories, digital libraries, and digital archives. I also covered how information organization affects the discovery and access to digital resources on the Internet. My research interests cover cataloging and classification, information organization, metadata, information retrieval, information architecture, digital collections, scholarly communication, social media, user interaction with information systems, and others.

7. I am very familiar with a library cataloging encoding standard known as the "Machine-Readable Cataloging" standard, also known as "MARC," which became the national standard for sharing bibliographic data in the United States by 1971 and the international standard by 1973. MARC is the primary communications protocol for the transfer and storage of bibliographic metadata in libraries. Experts in my field reasonably rely upon MARC records when forming their opinions.

8. A MARC record consists of several fields, each of which contains specific data about the work. Each field is identified by a standardized, unique, three-digit code corresponding to the type of data that follows. **Appendix B** is a true and correct copy of Parts VII to X of "Understanding MARC Bibliographic: Machine-Readable Cataloging" (http://www.loc.gov/marc/umb/um07to10.html) from the Library of Congress. *Understanding MARC Bibliographic* was originally published by the Follett Software Co. and copyrighted in 1988. Editions in 1989, 1990, 1994, 1998, 2000, 2003 and 2009 include changes. The latest edition was copyrighted in 2009. This document explains commonly-used MARC fields. For example, the personal author of the work is recorded in Field 100, the title is recorded in Field 245, publisher information is recorded in Field 260, the physical volume and characteristics of a publication are recorded in Field 300, and topical subjects are recorded in the 650 fields.

9. The Online Computer Library Center (OCLC) is the largest bibliographic network of the world, with more than 534 million records and thousands of member institutions (many of which are libraries of some type) in more than 100 countries. OCLC was founded in 1967 to promote and support library cooperation. According to the "Third Article, Amended Articles of Incorporation of OCLC Online Computer Library Center, Inc.," OCLC was created "to establish, maintain and operate a computerized library network and to promote the evolution of library use, of libraries themselves, and of librarianship, and to provide processes and products for the benefit of library users and libraries, including such objectives as increasing availability of library resources to individual library patrons and reducing the rate of rise of library per-unit costs, all for the fundamental public purpose of furthering ease of access to and use of the ever-expanding body of worldwide scientific, literary and educational knowledge and information" (source: https://www.oclc.org/content/dam/oclc/membership/articles-of-incorporation.pdf).

10. OCLC members can contribute original cataloging records in MARC to the system or derive cataloging records from existing records, an activity referred to as "copy cataloging." When an OCLC participating institution acquires a work, it can create an original MARC record for the work in OCLC's Connexion system (a system for catalogers to create and share MARC records), and the system will automatically generate a code for the date of record creation in the yymmdd format,

and the creating library's OCLC symbol is recorded in subfield "a" of the 040 field. Once the MARC record is in Connexion, it becomes available to other OCLC members for adoption to their local online catalogs (*i.e.*, copy cataloging).

11. After a MARC record is created in Connexion, it also becomes searchable and viewable on WorldCat, which is a free web portal for users to explore more than 10,000 libraries worldwide. The record in WorldCat, however, is not presented in MARC fields. Instead, the data elements are labeled to help users interpret the record. Thus, the information stored in MARC records in Connexion is available to the interested public through the user-friendly WorldCat web portal.

12. WorldCat (http://www.worldcat.org) is "the world's largest network of library content and services" and its features are summarized in "What is WorldCat" (http://www.worldcat.org/whatis/default.jsp). WorldCat began operation in January 1998 as a free web portal to more than 10,000 library collections worldwide. Through WorldCat, users can search for information resources in their local libraries and libraries around the world. WorldCat allows users to search for books, CDs, videos, and many new types of digital content, such as audiobooks, in many languages. Users can also retrieve research materials and article citations with links to their full text. After an item is retrieved, WorldCat helps users identify a library nearby that holds the item or all the libraries that hold the item. WorldCat is an efficient way to explore the content held by more than 10,000 libraries around the

world. WorldCat is publicly accessible, and there is no fee to perform a search on WorldCat.

13. Library online catalogs gained acceptance in the early 1980s and many libraries migrated their systems to the World Wide Web in the mid-1990s. Library online catalogs are based on MARC records that represent their collections in order to help the public understand what materials are publicly accessible in those libraries. Most libraries with online catalogs have made their catalogs freely available on the Web. These online catalogs offer user-friendly search interfaces. Strong user interest in keyword searches and the popularity of Google have led to the "googlization" of library search systems. As a result, many library catalogs now provide a single search box for users to conduct keyword searches, with additional support for searches by author, title, subject terms, and other data elements such as ISBN (International Standard Book Number). Library catalogs these days also offer features for users to narrow their search results by language, year, format, and other elements. Many libraries display MARC records on their online catalogs with labels for the data elements to help the public interpret MARC records. Many libraries also offer the option to display MARC records in MARC fields.

14. Libraries create MARC records for works they acquire, including books, serials, motion pictures, and publications in other formats. Monograph cataloging is fairly common in libraries, and most libraries make a newly cataloged

monograph available to the public soon after the cataloging work is completed, usually within a week. Libraries can create original cataloging records or use an existing record in OCLC to create a copy cataloging record. As soon as the cataloging record is completed, it is added to the library's online catalog for users. If the record is an original record, it is also entered into the OCLC system. If it is a copy cataloging record, the library's holding symbol is attached to the existing original record in the OCLC system to facilitate searching and interlibrary loan.

15. The cataloging of serials and the serial check-in process are discussed here to show how libraries usually provide access to newly received serial issues. According to the glossary of the *RDA: Resource Description and Access* cataloging standard, a serial is "a mode of issuance of a manifestation issued in successive parts, usually bearing numbering, that has no predetermined conclusion. A serial includes a periodical, monographic series, newspaper, etc." Because the publisher of a serial makes new issues of the serial available successively, a customary cataloging practice is to create one bibliographic record for the serial, and the serial record (encoded in MARC) typically provides information on the beginning date and frequency of the serial, not the dates of individual issues. In other words, libraries typically do not create MARC records for individual issues of a serial. Instead, they rely on a serial check-in system to track the receipt of new issues. A common check-in practice is to date stamp a new issue when it arrives. This practice has become

automated since the late 1990s, and libraries now vary in how they share the receipt date of a new serial issue with the public. Some libraries use a date stamp, some affix a label to indicate the receipt date, some pencil in the receipt date, and some do not provide the information to the public.

16. The serial check-in process usually takes less than an hour, and one of the steps involves placing a date stamp on the new issue to document the date the issue is checked in. After that, the holdings information of the serial is updated in the library's catalog so that users know which issues are available for request or access. After serial check-in is completed, the new issue is placed on the shelf with the previous issues of the serial. Libraries with a public periodical room typically place new issues in the periodical room for easy user access. Because information presented in serials often reflects latest discovery, a general practice of libraries is to make new issues of serials available for user access soon after they are checked in, usually within a week.

17. I am personally familiar with many online catalogs, databases, and search engines. In preparing for this declaration, I used the following authoritative information systems to search for records:

- Google Scholar (https://scholar.google.com)
- Internet Archive (https://archive.org)
- Online catalog of the British Library (http://explore.bl.uk/)

- Online catalog of the Library of Congress (https://catalog.loc.gov)
- WorldCat (https://www.worldcat.org)

**B. Scope of This Declaration**

18. I have been asked to offer opinions on the authenticity and public availability date of the following documents:

> Tahan, G., Rokach, L., & Shahar, Y., (2012, April), Mal-id: Automatic malware detection using common segment analysis and meta-features, *Journal of Machine Learning Research*, *13*(spring), pp. 949-979, **EX1003** ("*Tahan*");
>
> Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y., (2009), Improving malware detection by applying multi-inducer ensemble, *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483-1494, **EX1004** ("*Menahem*").

19. I am rendering my expert opinion on the authenticity of the document referenced herein and on whether it was published. I will provide my opinion on when and how this document was publicly disseminated or otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art, exercising reasonable diligence, could have located and retrieved this document.

20. I am informed by counsel that an item is considered authentic if there is sufficient evidence to support a finding that the item is what it is claimed to be. I am also informed that authenticity can be established based on the contents of the documents themselves, such as the appearance, content, substance, internal patterns, protocols followed or other distinctive characteristics of the item.

21. I am informed by counsel that a given reference is "published" if it is publicly accessible upon a satisfactory showing that such a document has been disseminated or otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art could locate it exercising reasonable diligence and obtain the document. I have also been informed by counsel that materials available in a library constitute "printed publications" if they are cataloged and indexed according to general library practices and protocols that make the references available and accessible to members of the interested public.

22. I have been informed by counsel that the '844 patent was filed in the United States on May 12, 2015, and that it claims priority to an U.S. Patent Application No. 14/709,875, filed 5/12/2015. I have assessed public accessibility of the documents referenced below as of this claimed priority date.

**C. Person of Ordinary Skills in the Arts**

23. I am told by counsel that the subject matter of this proceeding relates to methods or detecting threats in executable computer profiles.

24. I have been informed by counsel that a "person of ordinary skill in the art at the time of the inventions" (POSA) is a hypothetical person who is presumed to be familiar with the relevant field and its literature at the time of the inventions. This hypothetical person is also a person of ordinary creativity, capable of understanding the scientific principles and literature applicable to the pertinent field.

25. I am told by counsel that persons of ordinary skill in this subject matter or art would have had a bachelor's degree in computer science, or a related field, or an equivalent technical degree or equivalent work experience, and an additional one to two years of education or experience in the field of computer security topics such as malware/exploit detection and/or machine learning. More education can supplement practical experience and vice versa.

26. It is my opinion that such a person would have been actively engaged in learning about the field, possibly through formal instruction through the bibliographic resources. By at least May 12, 2015, such a person would have had access to a vast array of print and electronic resources, including at least the documents referenced below.

**D. Evidence Considered in Forming My Opinions**

27. In the preparation of this declaration, I have reviewed the documents referenced below and any other documents I reference herein, and each of these is a

type of material that experts in my field would reasonably rely upon when forming their opinions:

(1) The document referenced above in Section I.B;

(2) "JMLR Volume 13" website, archived by the Internet Archive on June 13, 2013, available at https://web.archive.org/web/20130613002437/http:/jmlr.org/papers/v13/, accessed and on February 16, 2023, **Appendix 1003A**;

(3) Abstract and copy of *Tahan*, archived by the Internet Archive on September 21, 2013, available at https://web.archive.org/web/20130921035310/http://jmlr.org/papers/v13/tahan12a.html and https://web.archive.org/web/20130921100447/http://jmlr.org/papers/volume13/tahan12a/tahan12a.pdf, accessed and on February 16, 2023, **Appendix 1003B**;

(4) Early citations to *Tahan*, obtained from Google Scholar, **Appendix 1003C;**

(5) Library copy of *Menahem*, obtained from the Library of Congress, **Appendix 1004A**;

(6) Bibliographic and MARC records for *Computational Statistics & Data Analysis* that contains *Menahem*, available at https://lccn.loc.gov/84642334 from the online catalog of the Library of Congress, accessed and obtained on February 17, 2023, **Appendix 1004B**;

(7) Library copy of the front matter of the issue of *Computational*

*Statistics & Data Analysis* that contains *Menahem*, obtained from the British Library, **Appendix 1004C**;

(8) Copy of *Menahem*, archived by the Internet Archive on December 3, 2011, available at http://web.archive.org/web/20111203120533/https://www.ise.bgu.ac.il/faculty/liorr/malens.pdf, accessed and obtained on February 17, 2023, **Appendix 1004D**;

(9) Selected citations to *Menahem*, obtained from Google Scholar, **Appendix 1004E**.

These records are identified and discussed in this declaration. Experts in the field would reasonably rely on the data described herein to form their opinions.

## II. Authenticity and Public Availability of *Tahan* (EX1003)

### A. Authentication

28. **Ex1003** is a true and correct copy of "Mal-id: Automatic malware detection using common segment analysis and meta-features," ("*Tahan*"), *Journal of Machine Learning Research*, *13*(spring), pp. 949-979, that I obtained from the journal website. When I began preparing this declaration, I searched Google Scholar for *Tahan* and a link from the article entry led me to the JMLR copy at https://www.jmlr.org/papers/volume13/tahan12a/tahan12a.pdf. This publisher copy is presented as **EX1003** in this declaration.

29. **EX1003** is a true and correct copy of *Tahan* obtained from the journal website. Page 1 shows the title, authors, abstract and keywords ("computer security,

malware detection, common segment analysis, supervised learning") of this article. The top shows the manuscript was submitted on August 11 and revised on November 11 of 2011, and the article was published on April 12 of 2012 in "Journal of Machine Learning Research 13 (2012) 949-979." The bottom of the page shows a "2012" copyright date, and the authors are the copyright holders. **EX1003** shows *Tahan* has a total of 31 pages, including eight tables, 10 figures (most of them color), and 41 references.

30. The JMLR copy of *Tahan* (**EX1003**) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Moreover, **EX1003** was found in the custody of the journal website, a place where, if authentic, it would likely be found. Accordingly, I see no reason to question the authenticity of this copy of *Tahan* (**EX1003**).

**B. Internet Archive Records**

31. To determine when *Tahan* became publicly accessible on the Internet, I searched the Internet Archive for records. **Appendix 1003A** is a true and correct copy of the JRML volume 13 website archived by the Internet Archive. I personally located, identified, and obtained this archived file. This is the type of material experts in my field would reasonably rely upon when forming their opinions.

32. Page 1 of **Appendix 1003A** is a screenshot of a Wayback Machine calendar to show the URL of the preserved website is http://jmlr.org/papers/v13/ and the website was archived 116 times from "10 Jun 2013" to "6 Dec 2022." Page 1 also shows this particular copy was archived on "JUN 13 2013" and is available at https://web.archive.org/web/20130613002437/http:/jmlr.org/papers/v13/. Internet Archive is a digital library that archives scholarly materials, cultural artifacts, and other web resources on the Internet and provides the public with free access to its collections. I have used the Internet Archive in the past and am familiar with their practice of assigning an extended URL to an archived resource in the format of http://www.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] to show the point of time the archived file was captured. The URL of this archived copy shows this website was archived on "20130613002437" (*i.e.*, June 13, 2013, 00 a.m., 24 minutes, 37 second). Because Internet Archive does not archive web resources as soon as they appear on the Internet, the archive date is not the first public availability date of an archived resource. Furthermore, a web resource will need to be available on the Internet before it can be archived. For these reasons, the archive date in the extended URL means that an archived copy of a web resource was available before, and definitely no later than, the archive date. It is therefore my opinion that this archived copy of this website would have been publicly available by June 13, 2013.

33. Pages 2 to 12 of **Appendix 1003A** are the PDF version of the archived website. Each entry represents a paper, including the title, authors, issue date, pagination, year of publication, and links to the abstract and the PDF of the paper. Page 4 shows the *Tahan* article was published in "13(Apr):949-979, 2012" and links to the abstract and the PDF version are provided.

34. **Appendix 1003B** is a true and correct copy of the abstract and the full text of *Tahan* archived by Internet Archive. I personally located, identified, and obtained this archived file. This is the type of material experts in my field would reasonably rely upon when forming their opinions.

35. Page 1 of **Appendix 1003B** is a screenshot of a Wayback Machine calendar to show the URL of the preserved abstract of *Tahan* is http://jmlr.org/papers/v13/tahan12a.html, and the abstract was archived 14 times from "21 Sep 2013" to "29 Oct 2020." The calendar shows this particular copy was archived on SEP 21 2013" and is available at http://web.archive.org/web/20130921035310/http://jmlr.org/papers/v13/tahan12a.htmlJul 2013." The URL of this archived copy shows this abstract was archived on "20130921035310" (*i.e.*, September 21, 2013, 3 a.m., 53 minutes, 10 seconds).

36. Page 2 of **Appendix 1003B** is a Wayback Machine calendar to shows the URL of the preserved article is http://jmlr.org/papers/volume13/tahan12a/tahan12a.pdf and the article was archived

21 times from "21 Sep 2013" to "25 Sep 2022." The calendar also shows this particular copy was archived on "SEP 21 2013" and is available at http://web.archive.org/web/20130921100447/http://jmlr.org/papers/volume13/tahan12a/tahan12a.pdf. The "20130921100447" means this article was archived on September 21, 2013, 10 a.m., 4 minutes, 47 seconds. Pages 3 to 33 of **Appendix 1003B** is the PDF version of *Tahan*. I have closely compared this copy with the publisher copy (**EX1003**) and found them to be the same.

37. Based on these archived records from the Internet Archive, volume 13 of JMLR made *Tahan* accessible from the website by June 13, 2013, when that website was archived, and the abstract and full text of *Tahan* became publicly accessible by September 21, 2013, when they were archived. It is therefore my opinion that the general public would have been able to access *Tahan* by June 13, 2013, and definitely no later than September 21, 2013, on the Internet.

**C. Usage Record**

38. Actual usage of a publication is reflected by the papers that make reference to it. The citation history on Google Scholar shows that *Tahan* has been cited 106 times, and eight citations were published in 2013. **Appendix 1003C** is a list I compiled from Google Scholar to demonstrate early usage. The earliest citations appeared in January 2013.

### D. Summary of My Opinion on *Tahan*

39. The publisher copy of *Tahan* (**EX1003**) is an authentic copy, the archived website of volume 13 of JRML (**Appendix 1003A**) shows that by June 13, 2013, visitors to the website would have learned that *Tahan* was published in April 2012, and they could follow the links to access the abstract and full text of this article. The archived copies of the abstract and full text of *Tahan* (**Appendix 1003B**) show that by September 21, 2013, *Tahan* was publicly accessible. Based on these records, it is my opinion that *Tahan* was publicly accessible as early as June 13, 2013, and no later than September 21, 2013, on the Internet. Citation history on Google Scholar shows eight citations were published in 2013 and the earliest citation appeared in January 2013.

## III. Authenticity and Public Availability of *Menahem* (EX1004)

### A. Authentication

40. **EX1004** is a true and correct copy of "Improving malware detection by applying multi-inducer ensemble," ("*Menahem*"), by Menahem et al., *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483-1494, that I obtained from the publisher Elsevier. When I began preparing this declaration I searched Google Scholar by the title of *Menahem* for records and a link from the entry led me to the publisher's webpage for *Menahem* at https://www.sciencedirect.com/science/article/abs/pii/S0167947308004763. I then

followed the PDF link in the record to obtain a copy. This publisher copy is presented as **EX1004** in this declaration.

41. **EX1004** is a true and correct copy of *Menahem* obtained from the publisher. Page 1 shows the title, authors, and abstract of this article, and also shows the manuscript was received "23 July 2008," accepted "4 October 2008," and became available online on "19 October 2008." This page shows *Menahem* appears in *Computational Statistics and Data Analysis* 53 (2009) 1483-1494, and is available at ScienceDirect, which is a platform of Elsevier to provide full text articles in science, technology, and other fields. The bottom of page shows *Menahem* appears in a journal whose ISSN (International Standard Serial Number) is "0167-9473" and the article has a "2008" copyright date, with Elsevier B.V. as the copyright holder, and the doi (digital object identifier) of *Menahem* is "10.1016/j.csda.2008.10.015." **EX1004** shows *Menahem* has a total of 12 pages, including seven equations, three figures (some color), ten tables, and 35 references.

42. To determine if *Menahem* was available at libraries, I searched WorldCat by the journal title and located a record at https://worldcat.org/title/962364724 that shows the Library of Congress and the British Library are two of the holdings libraries. I then checked the online catalog of the Library of congress, and the search results confirmed the holding information, so I obtained a scanned copy of *Menahem* via Wisconsin TechSearch (WTS), a

document delivery service based at the University of Wisconsin-Madison. This library copy is presented as **Appendix 1004A** in this declaration.

43. **Appendix 1004A** is a true and correct copy of *Menahem* obtained from the Library of Congress. Page 1 shows the covers and spine of volume 53 of *Computational Statistics & Data Analysis*, and the volume contains "Feb 15-Mar 15 2009, 4-5" that shows a call number of "QA 276.A1 C665 Set 1." Page 2 is the cover of the journal that identifies this issue as "Volume 53, Issue 4, 15 February 2009" and the ISSN as "0167-9473." The bottom of the page shows this journal is the official journal of iasc of isi, and iasc is the International Association for Statistical Computer, a section of the International Statistical Institute, and the journal is available online at ScienceDirect at www.sciencedirect.com. Page 3 is the issue cover that shows Elsevier of Amsterdam is the publisher. This page also carries a date stamp, but enlarged copies in pages 4 and 5 show that the stamp is not legible. Page 6 is the copyright page that shows a "2009" copyright date for this issue, and B.V. as the copyright holder. Pages 7 to 10 are the table of contents that shows *Menahem* begins on page 1483 and ends on page 1494. Pages 11 to 22 (internal pages 1483 to 1494) are the full text of *Menahem*. I have closely compared this copy of *Menahem* (**Appendix 1004A**) with the publisher copy (**EX1004**) and found them to be the same.

44. The Library of Congress copy of *Menahem* (**Appendix 1004**A) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is

not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alternations to the document. In addition, I have found this copy to be the same as the publisher copy. Moreover, **Appendix 1004**A was found in the custody of a national library, a place where, if authentic, it would likely be found. Accordingly, I see no reason to question the authenticity of this copy of *Menahem*.

**B. Library of Congress Records**

45. **Appendix 1004B** is a true and correct copy of the bibliographic and MARC records for *Computational Statistics and Data Analysis* that contains *Menahem*. I personally located, identified, and obtained these records from the online catalog of the Library of Congress. These are the type of records experts in my field would reasonably rely upon when forming their opinions.

46. The top of **Appendix 1004B** is the bibliographic record that identifies this publication as "periodical or newspaper" and shows the title, publishers, description, ISSN, and other bibliographic details. The "Item Availability" area shows the call number of this journal is "QA276.A1 C665 Set 1" and users can request access at the Jefferson or Adams Building Reading Rooms. The status of the journal issues is "Not Charged" (meaning available for user access), and the "Older receipts" field shows volume 53, no. 4 is included in the library's holdings. The record also shows unbound issues with the call number "QA276.A1 C665 Copy 1

Unbound issues" can be requested at the Newspaper & Current Periodical Reading Room in Madison Building "LM133".

47. Pages 6 to 10 are the MARC record. The first six digits of Field 008 show the MARC record was created on "830415" (*i.e.*, April 15, 1983), and the "c19839999" code after the creation date indicates this publication is a continuing resource that began publication in 1983 and has no end date of publication. Subfields "a" and "c" of Field 040 show "LUU" created the original record and subfield "d" shows "DLC" modified the original record to create a copy cataloging record later. "LUU" is the OCLC code for the library of Louisiana State University, and "DLC" is the OCLC library symbol for the Library of Congress, according to the *Directory of OCLC Members* (https://www.oclc.org/en/contacts/libraries.html). The serial MARC record has made *Computational Statistics & Data Analysis* that contains *Menahem* discoverable in the catalog of the Library of Congress.

48. Field 022 shows the ISSN of this journal is "0167-9473," Field 245 shows the journal title, Field 246 shows a variant title, and Fields 260 show the publishers. Field 362 shows the journal began publication in March 1983, and Fields 321 and Field 310 show the publication frequency changed over time. Field 530 indicates the journal is also "issued online via the World Wide Web" and Field 550 shows this journal has been "[t]he Official journal of International Association for Statistical Computer" since January 1994. Field 710 shows the parent organization is

included in the record to help. Users discover this journal.

49. Subjects of this journal are represented in Field 050 by a Library of Congress Classification (LCC) number of "QA276.A1" that represents journals on "Mathematical statistics" and in Field 082 by a Dewey Decimal Classification number of "519.5/05" that represents journal on "Statistical mathematics." Subjects are also represented in Fields 650 by three Library of Congress subject heading, with "Statistics," "Mathematical statistics," and "Mathematical statistics |x Data processing" ("|x" indicates a topical subdivision) as the main topics, each followed by "Periodicals," a form subdivision encoded in subfield "v", to indicate the main topics are treated in a journal.

50. This serial MARC record has made *Computational Statistics & Data Analysis* discoverable in the library's catalog. Interested users would have been able to conduct subject searches by the LCC number, the DDC number, and the Library of Congress subject headings to locate this journal. They would also have been able to search for it by its ISSN, title, variant title, and the parent organization of the journal.

51. Because the date stamp on the Library of Congress copy of *Menahem* is not legible, I obtained from the British Library the front matter of the journal issue containing *Menahem* to determine a date of public availability. **Appendix 1004C** is the copy obtained from the British Library. I personally located, identified and

obtained this copy. This is the type of material experts in my field would reasonably rely upon when forming their opinions.

52. **Appendix 1004C** is a true and correct copy of the front matter of volume 53, no. 4 of *Computational Statistics & Data Analysis*. Page 1 is the cover that is identical to the cover included in the Library of Congress copy of this issue (**Appendix 1004A**). The cover carries a date stamp indicating the receipt date is "24/02/09" (*i.e.*, February 24, 2009), and the call number of this journal is "3390.625000." Page 2 is the copyright page, and pages 3 to 6 are the table of contents. The British Library's copy of the front matter of vol. 53, issue 4 of this journal is identical to the front matter in the Library of Congress copy (**Appendix 1004A**). The date stamp in the British Library copy informs me that the library received this issue on February 24, 2009. As discussed above, because the publisher of a serial makes new issues of the serial available successively, a customary library cataloging practice is not to create MARC records for individual issues of a serial. Instead, libraries rely on a serial check-in system to track the receipt of new issues. The serial check-in process usually takes less than an hour, and one of the steps involves placing a date stamp on the new issue to document the date the issue is checked in. After that, the holdings information of the serial is updated in the library's catalog so that users know which issues are available for request or access. After serial check-in is completed, the new issue is placed on the shelf with the previous

issues of the serial. Because information presented in serials often reflects latest discovery, a general practice of libraries is to make new issues of serials available for user access soon after they are checked in, usually within a week. The date stamp of on the British Library copy (**Appendix 1004C**) informs my opinion that vol. 53, issue 4 of *Computational Statistics & Data Analysis* that contains *Menahem* was received by the library on February 24, 2009, and after the issue went through the serial check-in process, it would have been publicly accessible no later than March 3, 2009, at the library.

### C. Internet Archive Records

53. To determine if *Menahem* was available on the Internet, I searched the Internet Archive for records. **Appendix 1004D** is a true and correct copy of *Menahem*. I personally located, identified, and obtained this copy from the Internet Archive. This is the type of material experts in my field would reasonably rely upon when forming their opinions.

54. **Appendix 1004D** is a true and correct copy of *Menahem*. Page 1 is a screenshot of a Wayback Machine calendar to show the URL of the archived article is https://www.ise.bgu.ac.il/faculty/liorr/malens.pdf and the article was archived 12 times from "3 Dec 2011" to "2 Jan 2023." The calendar also shows this particular copy was archived on "DEC 03 2011" and is available at http://web.archive.org/web/20111203120533/https://www.ise.bgu.ac.il/faculty/liorr/

malens.pdf. As discussed above, the archive time in this extended URL means this copy was archived on "20111203120533" (*i.e.*, December 3, 2011, 12 p.m., 5 minutes, 33 seconds).

55. Pages 2 to 22 are the PDF version of *Menahem*. I have closely compared the archived copy (**Appendix 1004D**) with the publisher copy (**EX1004**) and found them to have the same content. The differences are that the archived copy shows five keywords provided by the authors, does not carry page numbers, does not show the corresponding author note, the ISSN of the journal or the doi of *Menahem*. In addition, the references are numbered in the archived copy.

56. The archived copy of *Menahem* (**Appendix 1004D**) informs my opinion that a copy of *Menahem* would have been publicly accessible no later than December 3, 2011, on the Internet.

**D. Internet Archive Affidavit (EX1019)**

57. My analysis of the archived copy of *Menahem* (**Appendix 1004D**) above is supported by the Declaration of Mr. Nathaniel E. Frank-White of the Internet Archive (**EX1019**) that I obtained from counsel. Mr. Frank-White states that "[t]he Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public" (Page 1, ¶ 2). He further explains that "[t]he Internet Archive assigns a URL on its site to the archived

files in the format http://www.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] aka an 'extended URL' … The date indicated by an extended URL applies to a preserved instance of a file for a given URL" (Page 1, ¶ 5). He then states that Exhibit B to his declaration "are true and accurate copies of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout" (Page 2, ¶ 7).

58. Mr. Frank-White's Exhibit B (**EX1019,** Pages 12 to 33) shows the URL and the archived copy of *Menahem* that are the same as the URL and the content in the archived copy discussed (**Appendix 1004D**). Exhibit B thus confirms that this archived copy includes a "true and accurate" copy "of the Internet Archive's records of the archived files" (**EX1019**, Page 1, ¶ 6) and supports my analysis that *Menahem* was indeed archived on December 3, 2011, by the Internet Archive.

### E. Citation Record

59. Actual usage of a publication is reflected by the papers that make reference to it. The citation history on Google Scholar shows *Menahem* has been cited at least 174 times. **Appendix 1004E** presents citations from 2009 to 2011 to demonstrate early usage. The earliest citation appeared in 2009.

### F. Summary of My Opinion on *Menahem*

60. The "24/02/09" date stamp on the British Library copy of the front matter of the journal issue containing *Menahem* (**Appendix 1004C**), the Library of

Congress copy of *Menahem* (**Appendix 1004A**), the bibliographic and MARC records for *Computational Statistics & Data Analysis* that contains *Menahem* (**Appendix 1004B**), and the archived copy of *Menahem* on Internet Archive (**Appendix 1004D**) that was archived on December 3, 2011, together with my knowledge of customary library cataloging and processing of serials, inform my opinion that *the journal issue containing Menahem* has been publicly accessible at the British Library no later than March 3, 2009, and the archived copy of *Menahem* would have been publicly accessible no later than December 3, 2011. Citation history shows the earliest citation was published in 2009.

## IV. Conclusion

61. In signing this declaration, I recognize that the declaration will be filed as evidence in a Petition for *Inter Partes* Review before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to deposition in the case. If deposition is required of me, I will appear at a reasonable time and place to be agreed upon.

62. I hereby declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Date: 3-3-2023 Executed: Ingrid Hsieh-Yee

Ingrid Hsieh-Yee, Ph.D.

# DECLARATION OF NATHANIEL E FRANK-WHITE

1. I am a Records Request Processor at the Internet Archive. I make this declaration of my own personal knowledge.

2. The Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public. The Internet Archive has partnered with and receives support from various institutions, including the Library of Congress.

3. The Internet Archive has created a service known as the Wayback Machine. The Wayback Machine makes it possible to browse more than 450 billion pages stored in the Internet Archive's web archive. Visitors to the Wayback Machine can search archives by URL (i.e., a website address). If archived records for a URL are available, the visitor will be presented with a display of available dates. The visitor may select one of those dates, and begin browsing an archived version of the Web. Links on archived files in the Wayback Machine point to other archived files (whether HTML pages or other file types), if any are found for the URL indicated by a given link. For instance, the Wayback Machine is designed such that when a visitor clicks on a hyperlink on an archived page that points to another URL, the visitor will be served the archived file found for the hyperlink's URL with the closest available date to the initial file containing the hyperlink.

4. The archived data made viewable and browseable by the Wayback Machine is obtained by use of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.

5. The Internet Archive assigns a URL on its site to the archived files in the format http://web.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] aka an "extended URL". Thus, the extended URL http://web.archive.org/web/19970126045828/http://www.archive.org/ would be the URL for the record of the Internet Archive home page HTML file (http://www.archive.org/) archived on January 26, 1997 at 4:58 a.m. and 28 seconds (1997/01/26 at 04:58:28). The date indicated by an extended URL applies to a preserved instance of a file for a given URL, but not necessarily to any other files linked therein. Thus, in the case of a page constituted by a primary HTML file and other separate files (e.g., files with images, audio, multimedia, design elements, or other embedded content) linked within that primary HTML file, the primary HTML file and the other files will each have their own respective extended URLs and may not have been archived on the same dates.

6. Attached hereto as Exhibit A are true and accurate copies of screenshots of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout.

7. Attached hereto as Exhibit B are true and accurate copies of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each file.

8. I declare under penalty of perjury that the foregoing is true and correct.

DATE: Febuary 24, 2023

Nathaniel E Frank-White

Nathaniel E Frank-White

# EXHIBIT A

https://web.archive.org/web/20130613002437/http:/jmlr.org/papers/v13/

INTERNET ARCHIVE WayBackMachine http://jmlr.org/papers/v13/ Go MAY JUN AUG 13 2012 2013 2014 About this capture
118 captures
10 Jun 2013 - 6 Dec 2022



# JMLR Volume 13

Distance Metric Learning with Eigenvalue Optimization
***Yiming Ying, Peng Li***; 13(Jan):1−26, 2012.
[abs][pdf]

Conditional Likelihood Maximisation: A Unifying Framework for Information Theoretic Feature Selection
***Gavin Brown, Adam Pocock, Ming-Jie Zhao, Mikel Luján***; 13(Jan):27−66, 2012.
[abs][pdf]

Plug-in Approach to Active Learning
***Stanislav Minsker***; 13(Jan):67−90, 2012.
[abs][pdf]

Refinement of Operator-valued Reproducing Kernels
***Haizhang Zhang, Yuesheng Xu, Qinghui Zhang***; 13(Jan):91−136, 2012.
[abs][pdf]

An Active Learning Algorithm for Ranking from Pairwise Preferences with an Almost Optimal Query Complexity
***Nir Ailon***; 13(Jan):137−164, 2012.
[abs][pdf]

Optimal Distributed Online Prediction Using Mini-Batches
***Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, Lin Xiao***; 13(Jan):165−202, 2012.
[abs][pdf]

Active Clustering of Biological Sequences
***Konstantin Voevodski, Maria-Florina Balcan, Heiko Röglin, Shang-Hua Teng, Yu Xia***; 13(Jan):203−225, 2012.
[abs][pdf]

Multi Kernel Learning with Online-Batch Optimization
***Francesco Orabona, Luo Jie, Barbara Caputo***; 13(Feb):227−253, 2012.
[abs][pdf]

Active Learning via Perfect Selective Classification
***Ran El-Yaniv, Yair Wiener***; 13(Feb):255−279, 2012.
[abs][pdf]

Random Search for Hyper-Parameter Optimization
***James Bergstra, Yoshua Bengio***; 13(Feb):281−305, 2012.
[abs][pdf]

Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics
***Michael U. Gutmann, Aapo Hyvärinen***; 13(Feb):307−361, 2012.
[abs][pdf]

Bounding the Probability of Error for High Precision Optical Character Recognition
***Gary B. Huang, Andrew Kae, Carl Doersch, Erik Learned-Miller***; 13(Feb):363−387, 2012.
[abs][pdf]

Minimax-Optimal Rates For Sparse Additive Models Over Kernel Classes Via Convex Programming
***Garvesh Raskutti, Martin J. Wainwright, Bin Yu***; 13(Feb):389−427, 2012.
[abs][pdf]

Online Learning in the Embedded Manifold of Low-rank Matrices
***Uri Shalit, Daphna Weinshall, Gal Chechik***; 13(Feb):429−458, 2012.
[abs][pdf]

Multi-Assignment Clustering for Boolean Data
***Mario Frank, Andreas P. Streich, David Basin, Joachim M. Buhmann***; 13(Feb):459−489, 2012.
[abs][pdf]

Eliminating Spammers and Ranking Annotators for Crowdsourced Labeling Tasks
***Vikas C. Raykar, Shipeng Yu***; 13(Feb):491−518, 2012.
[abs][pdf]

Metric and Kernel Learning Using a Linear Transformation
***Prateek Jain, Brian Kulis, Jason V. Davis, Inderjit S. Dhillon***; 13(Mar):519−547, 2012.
[abs][pdf]

MULTIBOOST: A Multi-purpose Boosting Package
***Djalel Benbouzid, Róbert Busa-Fekete, Norman Casagrande, François-David Collin, Balázs Kégl***; 13(Mar):549−553, 2012.
[abs][pdf] [code][mloss.org]

ML-Flex: A Flexible Toolbox for Performing Classification Analyses In Parallel
***Stephen R. Piccolo, Lewis J. Frey***; 13(Mar):555−559, 2012.
[abs][pdf] [code][sourceforge.net]

A Primal-Dual Convergence Analysis of Boosting
***Matus Telgarsky***; 13(Mar):561−606, 2012.

**Trend Micro, Inc.**
**EX1019-36**

[abs][pdf]

Non-Sparse Multiple Kernel Fisher Discriminant Analysis
***Fei Yan, Josef Kittler, Krystian Mikolajczyk, Atif Tahir***; 13(Mar):607−642, 2012.
[abs][pdf]

Learning Algorithms for the Classification Restricted Boltzmann Machine
***Hugo Larochelle, Michael Mandel, Razvan Pascanu, Yoshua Bengio***; 13(Mar):643−669, 2012.
[abs][pdf]

Structured Sparsity and Generalization
***Andreas Maurer, Massimiliano Pontil***; 13(Mar):671−690, 2012.
[abs][pdf]

A Case Study on Meta-Generalising: A Gaussian Processes Approach
***Grigorios Skolidis, Guido Sanguinetti***; 13(Mar):691−721, 2012.
[abs][pdf]

A Kernel Two-Sample Test
***Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, Alexander Smola***; 13(Mar):723−773, 2012.
[abs][pdf]

GPLP: A Local and Parallel Computation Toolbox for Gaussian Process Regression
***Chiwoo Park, Jianhua Z. Huang, Yu Ding***; 13(Mar):775−779, 2012.
[abs][pdf] [code][mloss.org]

Exact Covariance Thresholding into Connected Components for Large-Scale Graphical Lasso
***Rahul Mazumder, Trevor Hastie***; 13(Mar):781−794, 2012.
[abs][pdf]

Algorithms for Learning Kernels Based on Centered Alignment
***Corinna Cortes, Mehryar Mohri, Afshin Rostamizadeh***; 13(Mar):795−828, 2012.
[abs][pdf]

Causal Bounds and Observable Constraints for Non-deterministic Models
***Roland R. Ramsahai***; 13(Mar):829−848, 2012.
[abs][pdf]

NIMFA : A Python Library for Nonnegative Matrix Factorization
***Marinka Žitnik, Blaž Zupan***; 13(Mar):849−853, 2012.
[abs][pdf] [code][mloss.org]

Algebraic Geometric Comparison of Probability Distributions
***Franz J. Király, Paul von Bünau, Frank C. Meinecke, Duncan A.J. Blythe, Klaus-Robert Müller***; 13(Mar):855−903, 2012.
[abs][pdf]

Stability of Density-Based Clustering
***Alessandro Rinaldo, Aarti Singh, Rebecca Nugent, Larry Wasserman***; 13(Apr):905−948, 2012.
[abs][pdf]

Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features
***Gil Tahan, Lior Rokach, Yuval Shahar***; 13(Apr):949−979, 2012.
[abs][pdf]

Sampling Methods for the Nyström Method
***Sanjiv Kumar, Mehryar Mohri, Ameet Talwalkar***; 13(Apr):981−1006, 2012.
[abs][pdf]

Positive Semidefinite Metric Learning Using Boosting-like Algorithms
***Chunhua Shen, Junae Kim, Lei Wang, Anton van den Hengel***; 13(Apr):1007−1036, 2012.
[abs][pdf]

Consistent Model Selection Criteria on High Dimensions
***Yongdai Kim, Sunghoon Kwon, Hosik Choi***; 13(Apr):1037−1057, 2012.
[abs][pdf]

The huge Package for High-dimensional Undirected Graph Estimation in R
***Tuo Zhao, Han Liu, Kathryn Roeder, John Lafferty, Larry Wasserman***; 13(Apr):1059−1062, 2012.
[abs][pdf] [code][cran.r-project.org]

Analysis of a Random Forests Model
***Gérard Biau***; 13(Apr):1063−1095, 2012.
[abs][pdf]

Towards Integrative Causal Analysis of Heterogeneous Data Sets and Studies
***Ioannis Tsamardinos, Sofia Triantafillou, Vincenzo Lagani***; 13(Apr):1097−1157, 2012.
[abs][pdf]

Hope and Fear for Discriminative Training of Statistical Translation Models
***David Chiang***; 13(Apr):1159−1187, 2012.
[abs][pdf]

A Multi-Stage Framework for Dantzig Selector and LASSO
***Ji Liu, Peter Wonka, Jieping Ye***; 13(Apr):1189−1219, 2012.
[abs][pdf]

A Geometric Approach to Sample Compression

A Geometric Approach to Sample Compression
***Benjamin I.P. Rubinstein, J. Hyam Rubinstein***; 13(Apr):1221−1261, 2012.
[abs][pdf]

Minimax Manifold Estimation
***Christopher Genovese, Marco Perone-Pacifico, Isabella Verdinelli, Larry Wasserman***; 13(May):1263−1291, 2012.
[abs][pdf]

Query Strategies for Evading Convex-Inducing Classifiers
***Blaine Nelson, Benjamin I. P. Rubinstein, Ling Huang, Anthony D. Joseph, Steven J. Lee, Satish Rao, J. D. Tygar***; 13(May):1293−1332, 2012.
[abs][pdf]

Transfer in Reinforcement Learning via Shared Features
***George Konidaris, Ilya Scheidwasser, Andrew Barto***; 13(May):1333−1371, 2012.
[abs][pdf]

On Ranking and Generalization Bounds
***Wojciech Rejchel***; 13(May):1373−1392, 2012.
[abs][pdf]

Feature Selection via Dependence Maximization
***Le Song, Alex Smola, Arthur Gretton, Justin Bedo, Karsten Borgwardt***; 13(May):1393−1434, 2012.
[abs][pdf]

Structured Sparsity via Alternating Direction Methods
***Zhiwei Qin, Donald Goldfarb***; 13(May):1435−1468, 2012.
[abs][pdf]

Activized Learning: Transforming Passive to Active with Improved Label Complexity
***Steve Hanneke***; 13(May):1469−1587, 2012.
[abs][pdf]

A Model of the Perception of Facial Expressions of Emotion by Humans: Research Overview and Perspectives
***Aleix Martinez, Shichuan Du***; 13(May):1589−1608, 2012.
[abs][pdf]

A Unifying Probabilistic Perspective for Spectral Dimensionality Reduction: Insights and New Models
***Neil D. Lawrence***; 13(May):1609−1638, 2012.
[abs][pdf]

Mixability is Bayes Risk Curvature Relative to Log Loss
***Tim van Erven, Mark D. Reid, Robert C. Williamson***; 13(May):1639−1663, 2012.
[abs][pdf]

Restricted Strong Convexity and Weighted Matrix Completion: Optimal Bounds with Noise
***Sahand Negahban, Martin J. Wainwright***; 13(May):1665−1697, 2012.
[abs][pdf]

glm-ie: Generalised Linear Models Inference & Estimation Toolbox
***Hannes Nickisch***; 13(May):1699−1703, 2012.
[abs][pdf] [code][mloss.org]

Manifold Identification in Dual Averaging for Regularized Stochastic Online Learning
***Sangkyun Lee, Stephen J. Wright***; 13(Jun):1705−1744, 2012.
[abs][pdf]

Variational Multinomial Logit Gaussian Process
***Kian Ming A. Chai***; 13(Jun):1745−1808, 2012.
[abs][pdf]

Entropy Search for Information-Efficient Global Optimization
***Philipp Hennig, Christian J. Schuler***; 13(Jun):1809−1837, 2012.
[abs][pdf]

Estimation and Selection via Absolute Penalized Convex Minimization And Its Multistage Adaptive Applications
***Jian Huang, Cun-Hui Zhang***; 13(Jun):1839−1864, 2012.
[abs][pdf]

Regularization Techniques for Learning with Matrices
***Sham M. Kakade, Shai Shalev-Shwartz, Ambuj Tewari***; 13(Jun):1865−1890, 2012.
[abs][pdf]

Confidence-Weighted Linear Classification for Text Categorization
***Koby Crammer, Mark Dredze, Fernando Pereira***; 13(Jun):1891−1926, 2012.
[abs][pdf]

Integrating a Partial Model into Model Free Reinforcement Learning
***Aviv Tamar, Dotan Di Castro, Ron Meir***; 13(Jun):1927−1966, 2012.
[abs][pdf]

Jstacs: A Java Framework for Statistical Analysis and Classification of Biological Sequences
***Jan Grau, Jens Keilwagen, André Gohr, Berit Haldemann, Stefan Posch, Ivo Grosse***; 13(Jun):1967−1971, 2012.
[abs][pdf] [code][mloss.org]

Variable Selection in High-dimensional Varying-coefficient Models with Global Optimality
***Lan Xue, Annie Qu***; 13(Jun):1973−1998, 2012.
[abs][pdf]

An Improved GLMNET for L1-regularized Logistic Regression
***Guo-Xun Yuan, Chia-Hua Ho, Chih-Jen Lin***; 13(Jun):1999−2030, 2012.
[abs][pdf]

EP-GIG Priors and Applications in Bayesian Sparse Learning
***Zhihua Zhang, Shusen Wang, Dehua Liu, Michael I. Jordan***; 13(Jun):2031−2061, 2012.
[abs][pdf]

Pattern for Python
***Tom De Smedt, Walter Daelemans***; 13(Jun):2063−2067, 2012.
[abs][pdf] [code][mloss.org]

Optimistic Bayesian Sampling in Contextual-Bandit Problems
***Benedict C. May, Nathan Korda, Anthony Lee, David S. Leslie***; 13(Jun):2069−2106, 2012.
[abs][pdf]

A Comparison of the Lasso and Marginal Regression
***Christopher R. Genovese, Jiashun Jin, Larry Wasserman, Zhigang Yao***; 13(Jun):2107−2143, 2012.
[abs][pdf]

On the Necessity of Irrelevant Variables
***David P. Helmbold, Philip M. Long***; 13(Jul):2145−2170, 2012.
[abs][pdf]

DEAP: Evolutionary Algorithms Made Easy
***Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, Christian Gagné***; 13(Jul):2171−2175, 2012.
[abs][pdf] [code][deap.gel.ulaval.ca]

An Introduction to Artificial Prediction Markets for Classification
***Adrian Barbu, Nathan Lay***; 13(Jul):2177−2204, 2012.
[abs][pdf]

Sign Language Recognition using Sub-Units
***Helen Cooper, Eng-Jon Ong, Nicolas Pugeault, Richard Bowden***; 13(Jul):2205−2231, 2012.
[abs][pdf]

A Topic Modeling Toolbox Using Belief Propagation
***Jia Zeng***; 13(Jul):2233−2236, 2012.
[abs][pdf] [code][mloss.org]

MedLDA: Maximum Margin Supervised Topic Models
***Jun Zhu, Amr Ahmed, Eric P. Xing***; 13(Aug):2237−2278, 2012.
[abs][pdf]

Pairwise Support Vector Machines and their Application to Large Scale Problems
***Carl Brunner, Andreas Fischer, Klaus Luig, Thorsten Thies***; 13(Aug):2279−2292, 2012.
[abs][pdf]

High-Dimensional Gaussian Graphical Model Selection: Walk Summability and Local Separation Criterion
***Animashree Anandkumar, Vincent Y.F. Tan, Furong Huang, Alan S. Willsky***; 13(Aug):2293−2337, 2012.
[abs][pdf]

A Local Spectral Method for Graphs: With Applications to Improving Graph Partitions and Exploring Data Graphs Locally
***Michael W. Mahoney, Lorenzo Orecchia, Nisheeth K. Vishnoi***; 13(Aug):2339−2365, 2012.
[abs][pdf]

Multi-Target Regression with Rule Ensembles
***Timo Aho, Bernard Ženko, Sašo Džeroski, Tapio Elomaa***; 13(Aug):2367−2407, 2012.
[abs][pdf]

Characterization and Greedy Learning of Interventional Markov Equivalence Classes of Directed Acyclic Graphs
***Alain Hauser, Peter Bühlmann***; 13(Aug):2409−2464, 2012.
[abs][pdf]

On the Convergence Rate of $l_p$-Norm Multiple Kernel Learning
***Marius Kloft, Gilles Blanchard***; 13(Aug):2465−2502, 2012.
[abs][pdf]

Trading Regret for Efficiency: Online Convex Optimization with Long Term Constraints
***Mehrdad Mahdavi, Rong Jin, Tianbao Yang***; 13(Sep):2503−2528, 2012.
[abs][pdf]

Robust Kernel Density Estimation
***JooSeuk Kim, Clayton D. Scott***; 13(Sep):2529−2565, 2012.
[abs][pdf]

Nonparametric Guidance of Autoencoder Representations using Label Information
***Jasper Snoek, Ryan P. Adams, Hugo Larochelle***; 13(Sep):2567−2588, 2012.
[abs][pdf]

Finding Recurrent Patterns from Continuous Sign Language Sentences for Automated Extraction of Signs
***Sunita Nayak, Kester Duncan, Sudeep Sarkar, Barbara Loeding***; 13(Sep):2589−2615, 2012.
[abs][pdf]

Static Prediction Games for Adversarial Learning Problems
***Michael Brückner, Christian Kanzow, Tobias Scheffer***; 13(Sep):2617−2654, 2012.

[abs][pdf]

Selective Sampling and Active Learning from Single and Multiple Teachers
***Ofer Dekel, Claudio Gentile, Karthik Sridharan***; 13(Sep):2655−2697, 2012.
[abs][pdf]

PREA: Personalized Recommendation Algorithms Toolkit
***Joonseok Lee, Mingxuan Sun, Guy Lebanon***; 13(Sep):2699−2703, 2012.
[abs][pdf] [code][mloss.org]

Coherence Functions with Applications in Large-Margin Classification Methods
***Zhihua Zhang, Dehua Liu, Guang Dai, Michael I. Jordan***; 13(Sep):2705−2734, 2012.
[abs][pdf]

Linear Regression With Random Projections
***Odalric-Ambrym Maillard, Rémi Munos***; 13(Sep):2735−2772, 2012.
[abs][pdf]

Multi-task Regression using Minimal Penalties
***Matthieu Solnon, Sylvain Arlot, Francis Bach***; 13(Sep):2773−2812, 2012.
[abs][pdf]

A Unified View of Performance Metrics: Translating Threshold Choice into Expected Classification Loss
***José Hernández-Orallo, Peter Flach, Cèsar Ferri***; 13(Oct):2813−2869, 2012.
[abs][pdf]

Local and Global Scaling Reduce Hubs in Space
***Dominik Schnitzer, Arthur Flexer, Markus Schedl, Gerhard Widmer***; 13(Oct):2871−2902, 2012.
[abs][pdf]

Online Submodular Minimization
***Elad Hazan, Satyen Kale***; 13(Oct):2903−2922, 2012.
[abs][pdf]

Efficient Methods for Robust Classification Under Uncertainty in Kernel Matrices
***Aharon Ben-Tal, Sahely Bhadra, Chiranjib Bhattacharyya, Arkadi Nemirovski***; 13(Oct):2923−2954, 2012.
[abs][pdf]

Facilitating Score and Causal Inference Trees for Large Observational Studies
***Xiaogang Su, Joseph Kang, Juanjuan Fan, Richard A. Levine, Xin Yan***; 13(Oct):2955−2994, 2012.
[abs][pdf]

Oger: Modular Learning Architectures For Large-Scale Sequential Processing
***David Verstraeten, Benjamin Schrauwen, Sander Dieleman, Philemon Brakel, Pieter Buteneers, Dejan Pecevski***; 13(Oct):2995−2998, 2012.
[abs][pdf] [code][mloss.org]

Multi-Instance Learning with Any Hypothesis Class
***Sivan Sabato, Naftali Tishby***; 13(Oct):2999−3039, 2012.
[abs][pdf]

Finite-Sample Analysis of Least-Squares Policy Iteration
***Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos***; 13(Oct):3041−3074, 2012.
[abs][pdf]

Discriminative Hierarchical Part-based Models for Human Parsing and Action Recognition
***Yang Wang, Duan Tran, Zicheng Liao, David Forsyth***; 13(Oct):3075−3102, 2012.
[abs][pdf]

Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training
***Zhuang Wang, Koby Crammer, Slobodan Vucetic***; 13(Oct):3103−3131, 2012.
[abs][pdf]

Bayesian Mixed-Effects Inference on Classification Performance in Hierarchical Data Sets
***Kay H. Brodersen, Christoph Mathys, Justin R. Chumbley, Jean Daunizeau, Cheng Soon Ong, Joachim M. Buhmann, Klaas E. Stephan***; 13(Nov):3133−3176, 2012.
[abs][pdf]

Quantum Set Intersection and its Application to Associative Memory
***Tamer Salman, Yoram Baram***; 13(Nov):3177−3206, 2012.
[abs][pdf]

Dynamic Policy Programming
***Mohammad Gheshlaghi Azar, Vicenç Gómez, Hilbert J. Kappen***; 13(Nov):3207−3245, 2012.
[abs][pdf]

Sally: A Tool for Embedding Strings in Vector Spaces
***Konrad Rieck, Christian Wressnegger, Alexander Bikadorov***; 13(Nov):3247−3251, 2012.
[abs][pdf] [code][mloss.org]

Linear Fitted-Q Iteration with Multiple Reward Functions
***Daniel J. Lizotte, Michael Bowling, Susan A. Murphy***; 13(Nov):3253−3295, 2012.
[abs][pdf]

Human Gesture Recognition on Product Manifolds
***Yui Man Lui***; 13(Nov):3297−3321, 2012.
[abs][pdf]

Large-scale Linear Support Vector Regression
***Chia-Hua Ho, Chih-Jen Lin***; 13(Nov):3323−3348, 2012.
[abs][pdf]

Sparse and Unique Nonnegative Matrix Factorization Through Data Preprocessing
***Nicolas Gillis***; 13(Nov):3349−3386, 2012.
[abs][pdf]

Learning Linear Cyclic Causal Models with Latent Variables
***Antti Hyttinen, Frederick Eberhardt, Patrik O. Hoyer***; 13(Nov):3387−3439, 2012.
[abs][pdf]

Iterative Reweighted Algorithms for Matrix Rank Minimization
***Karthik Mohan, Maryam Fazel***; 13(Nov):3441−3473, 2012.
[abs][pdf]

Fast Approximation of Matrix Coherence and Statistical Leverage
***Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, David P. Woodruff***; 13(Dec):3475−3506, 2012.
[abs][pdf]

PAC-Bayes Bounds with Data Dependent Priors
***Emilio Parrado-Hernández, Amiran Ambroladze, John Shawe-Taylor, Shiliang Sun***; 13(Dec):3507−3531, 2012.
[abs][pdf]

DARWIN: A Framework for Machine Learning and Computer Vision Research and Development
***Stephen Gould***; 13(Dec):3533−3537, 2012.
[abs][pdf] [code][mloss.org]

Regularized Bundle Methods for Convex and Non-Convex Risks
***Trinh Minh Tri Do, Thierry Artières***; 13(Dec):3539−3583, 2012.
[abs][pdf]

Learning Symbolic Representations of Hybrid Dynamical Systems
***Daniel L. Ly, Hod Lipson***; 13(Dec):3585−3618, 2012.
[abs][pdf]

SVDFeature: A Toolkit for Feature-based Collaborative Filtering
***Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, Yong Yu***; 13(Dec):3619−3622, 2012.
[abs][pdf] [code][mloss.org]

Smoothing Multivariate Performance Measures
***Xinhua Zhang, Ankan Saha, S.V.N. Vishwanathan***; 13(Dec):3623−3680, 2012.
[abs][pdf]

Security Analysis of Online Centroid Anomaly Detection
***Marius Kloft, Pavel Laskov***; 13(Dec):3681−3724, 2012.
[abs][pdf]

Exploration in Relational Domains for Model-based Reinforcement Learning
***Tobias Lang, Marc Toussaint, Kristian Kersting***; 13(Dec):3725−3768, 2012.
[abs][pdf]

# EXHIBIT B

http://web.archive.org/web/20111203120533/https://www.ise.bgu.ac.il/faculty/liorr/malens.pdf

# Improving Malware Detection by Applying Multi-Inducer Ensemble

Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici

Duetsche Telekom Laboratories at Ben-Gurion University of the Negev,
Be'er Sheva, 84105 Israel
{eitanme, shabtaia, liorrk, elovici}@bgu.ac.il

**Abstract.** Detection of malicious software (malware) using machine learning methods has been explored extensively to enable fast detection of new released malware. The performance of these classifiers depends on the induction algorithms being used. In order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual classifiers into one final result to achieve overall higher detection accuracy. In this paper we evaluate several combining methods using five different base inducers (C4.5 Decision Tree, Naïve Bayes, KNN, VFI and OneR) on five malware datasets. The main goal is to find the best combining method for the task of detecting malicious files in terms of accuracy, AUC and execution time.

**Keywords:** Classification Algorithms, Malicious Code, Ensemble learning, Decision Tree, Naıve Bayes classifier

## 1 Introduction

Modern computer and communication infrastructure are highly susceptible to various types of attacks. A common way of launching these attacks is using malware (malicious software) such as worm, virus, Trojan or spyware [Kienzle et al., 2003; Heidrai, 2004]. A spreading malicious software might generate great damage to private users, commercial companies and governments. A single malware in a computer, which is a part of a computer network, can result in the loss, or unauthorized utilization or modification, of large amounts of data and cause users to question the reliability of all of the information on the network.

Detection of known malware is commonly performed by anti-virus tools. These tools detect the known malware using signature detection methods [White, 1999; Dikinson, 2005]. When a new malware is released, the anti-virus vendors need to catch an instance of the new malware, analyze it, create a new signature and update their clients. During the period between the appearance of a new unknown malware and the update of the signature-base of the anti-virus clients, millions of computers are vulnerable to the new malware. Therefore, although signature-based detection methods eventually yield high detection rate, they cannot cope with new, unseen before, malware [White, 1998]. High-speed communication channels enable malware

to propagate and infect hosts quickly and so it is essential to detect and eliminate new (unknown) spreading malware at early stages of the attack.

Recent studies have proposed methods for detecting malicious executables using Machine Learning (ML) techniques [Kabiri et al., 2005]. Given a training set of malicious and benign executables binary code, a classifier is trained to identify and classify unknown malicious executables as being malicious. Some of the methods are inspired by text categorization techniques, in which words are analogous to sequences in the binary code. Malicious and benign files are represented by a vector of features, extracted from the PE-header and the binary code of the executable. These files are used to train a classifier. During the detection phase, based on the generalization capability of the classifier, an unknown file can be classified as malicious or benign. Some of the studies applying machine learning methods on content-based features are mentioned below.

In [Schultz et al., 2001], ML techniques were applied for detection of unknown malicious code based on binary code content. Three different feature extraction methods are used: program header, printable strings features and byte Sequences features, on which they applied four classifiers: Signature-based method (anti-virus), Ripper – a rule based learner, Naïve Bayes and Multi Naïve Bayes. The experiments conducted using 4,301 programs in Win32 PE format (3,301 malicious programs and 1000 benign programs) and the results show that all ML methods outperform the signature-based classifier.

The study in [Abou-Assaleh et al., 2004] used *n*-gram features extracted from the binary code of the file. The *L* most frequent *n*-grams of each class in the training set are chosen and represent the profile of the class. A new instance is associated to a class with the closest profile using KNN algorithm. The experiment conducted on a small set of 65 malicious and benign files achieved 98% accuracy. Caia et al. (2007) and Moskovitch et al. (2008) have compared different feature selection methods for building a single classifier.

Kolter and Maloof (2006) extracted *n*-grams from the executables. Most relevant *n*-gram attributes were chosen using Info-Grain ratio and the data-sets were evaluated using several inducers: IBK, Naive Bayes, Support Vector Machines (SVMs), Decision Trees, Boosted Naive Bayes, Boosted SVMs, and Boosted Decision Trees algorithms. Since costs of misclassification error are usually unknown, the methods were evaluated using receiver operating characteristic (ROC) analysis. When classifying new instances into two optional classes 'Benign' and 'Malicious', Boosted decision trees outperformed other methods with an area under the ROC curve of 0.996. When classifying new instances into multi-class (such as Virus, Backdoor etc.), the area under the ROC curve reaches an average of 0.9.

By its nature, the measured performance (e.g., accuracy) of a machine learning classifier is influenced by two main factors: (1) the inducer algorithm that is used to generate the classifier (e.g., Artificial Neural Network, Decision Tree, Naïve Bayes); and (2) the type of features that are used to represent the instances. Thus, various types of classifiers have different “inductive biases”. When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently. It might be the case that a certain classifier will specialized in classifying instances into a sub-set of classes (for example, a classifier that classifies an executable file as benign or Worm with

high accuracy but with low accuracy when classifying into other malicious classes such as Trojan and Virus).

The outcome of this observation is that one should use multiple different classifiers that will classify a new instance predicting its real class (like a group of experts) and then combining all the results, using an intelligent combination method, into one final result. Hopefully, this "super-classifier" will outperform each of the individual classifiers and classify new instances with higher accuracy by learning the strength and weakness of each classifier.

In this paper we introduce three main contributions: (1) we show that multi-inducer ensembles are capable to detect malwares; (2) we introduce an innovative combining method, called Troika [Menahem, 2008] which extends Stacking and show it superiority in the malware detection tasks; and (3) we present empirical results from an extensive real world study of various malwares using different types of features.

The rest of the paper is organized as follows. In section 2 we describe related work in the field of ensemble application in the security domain. Section 3 presents the experimental setup and in section 4 we present the evaluation results. Conclusions and future work are discussed in section 5.

## 2 Related Work

In this section we present studies in which ensemble methods are used to aid in malicious code detection and in intrusion detection. We also describe the six ensemble methods used in our experiments.

### 2.1 Malware Detection Using Ensemble Methods

The main idea of an ensemble methodology is to combine a set of classifiers in order to obtain a better composite global classifier. Ensemble methodology imitates our second nature to seek several opinions before making any crucial decision. We weigh the individual opinions, and combine them to reach a final decision. Recent studies evaluated the improvement in performance (in terms of accuracy, false positive rate and area under the ROC graph) when using ensemble algorithms to combine the individual classifiers.

Several researchers examined the applicability of ensemble methods in detecting malicious code. Zhang et. al., (2007) classifies new malicious code based on *n*-gram features extracted from binary code of the file. First, *n*-gram based features are extracted and the best *n*-grams are chosen by applying the Information-Gain feature selection method. Then, probabilistic neural network (PNN) is used to construct several classifiers for detection. Finally, the individual decisions of the PNN classifiers are combined by using the Dempster-Shafer theory to create the combining rules. The method was evaluated on a set of malicious executables that were downloaded from the website VX Heavens (http://www.vx.netlux.org) and clean programs gathered from a Windows 2000 server machine: total of 423 benign programs and 450 malicious executable files (150 Viruses, 150 Trojans and 150 Worms) all in Windows PE format. The results show a better ROC curve for the

ensemble of PNNs when compared to the individual best PNN of each of the three malware classes.

Ensemble of classifiers was evaluated on intrusion detection tasks as well. Mukkamalaa et al., (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detected intrusions within network traffic. The performance of the ensemble results were compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS) and it was evaluated using DARPA data-set which is considered as an IDS evaluation benchmark. The classifiers were trained to detect normal traffic as well as four classes of attacks (probing, DOS, user to root and remote to user). It is shown that the simple majority voting schema improves the accuracy of the detection.

The ensemble approach in [Kelly et al., 2006] was evaluated using two anomaly network intrusion detection systems, LERAD [Mahoney, 2003] and PAYL [Wang and Stolfo, 2004]. LERAD's normal behavior is composed of rule sets of expected (normal) user behavior, and PAYL uses byte distributions derived from normal packets. Based on the classification of the two systems as produced on a training-set, an ensemble probability classification matrix (EPCM) is generated using conditional probability. The method was evaluated on DARPA data-set and results show an increase in the rate of detecting attacks and the accuracy in determining their exact classes.

In order to make the ensemble more effective, there should be some sort of diversity between the classifiers [Kuncheva, 2005]. In Multi-Inducer strategy, diversity is obtained by using different types of inducers. Each inducer contains an explicit or implicit bias that leads it to prefer certain generalizations over others. Ideally, this multi-inducer strategy would always perform as well as the best of its ingredients. Even more ambitiously, there is hope that this combination of paradigms might produce synergistic effects, leading to levels of accuracy that neither atomic approach by itself would be able to achieve.

In this study we compare several methods for combining various classifiers in order to understand which of these methods (if any) best improves the detection accuracy.

### 2.2 Methods for Combining Ensemble's Members

In the *Majority Voting* combining scheme [Kittler et al., 1998], a classification of an unlabeled instance is performed according to the class that obtains the highest number of votes (the most frequent vote). This method is also known as the plurality vote (PV) or the basic ensemble method (BEM). This approach has frequently been used as a combining method for comparing to newly proposed methods. Mathematically it can be written as:

$$class(x) = \underset{c_i \in dom(y)}{\arg\max}\left(\sum_k g(y_k(x), c_i)\right) \quad \textbf{(1)}$$

where $y_k(x)$ is the classification of the $k^{\text{th}}$ classifier and $g(y,c)$ is an indicator function defined as:

$$g(y,c)=\begin{cases}1, y=c\\ 0, y\neq c\end{cases} \quad \textbf{(2)}$$

Note that in case of a probabilistic classifier, the crisp classification $y_k(x)$ is usually obtained as follows:

$$y_k(x)=\underset{c_i\in dom(y)}{\arg\max}\,\hat{P}_{M_k}(y=c_i\mid x) \quad \textbf{(3)}$$

where $M_k$ denotes classifier $k$ and $\hat{P}_{M_k}(y=c_i\mid x)$ denotes the probability of $y$ obtaining the value $c$ given an instance $x$.

In *Performance Weighting*, the weight of each classifier is set proportional to its accuracy performance on a validation set [Opitz and Shavlik, 1996]:

$$\alpha_i=\frac{(1-E_i)}{\sum_{j=1}^{t}(1-E_i)} \quad \textbf{(4)}$$

where $E_i$ is a normalization factor which is based on the performance evaluation of classifier $i$ on a validation set.

The idea of *Distribution Summation* combining method is to sum up the conditional probability vector obtained from each classifier [Clark and Boswell, 1991]. The selected class is chosen according to the highest value in the total vector. Mathematically, it can be written as:

$$Class(x)=\underset{c_i\in dom(y)}{\arg\max}\sum_k \hat{P}_{M_k}(y=c_i\mid x) \quad \textbf{(5)}$$

In the *Bayesian Combination* method the weight associated with each classifier is the posterior probability of the classifier given the training set [Buntine, 1990].

$$Class(x)=\underset{c_i\in dom(y)}{\arg\max}\sum_k P(M_k\mid S)\cdot\hat{P}_{M_k}(y=c_i\mid x) \quad \textbf{(6)}$$

where $P(M_k|S)$ denotes the probability that the classifier $M_k$ is correct given the training set $S$. The estimation of $P(M_k|S)$ depends on the classifier's representation. To estimate this value for decision trees the reader is referred to [Buntine, 1990].

Using Bayes' rule, one can extend the *Naive Bayes* idea for combining various classifiers [John and Langley, 1995]:

$$Class(x)=\underset{\substack{c_j\in dom(y)\\ \hat{P}(y=c_j)>0}}{\arg\max}\,\hat{P}(y=c_j)\cdot\prod_{k=1}\frac{\hat{P}_{M_k}(y=c_j\mid x)}{\hat{P}(y=c_j)} \quad \textbf{(7)}$$

*Stacking* is a technique whose purpose is to achieve the highest generalization accuracy [Wolpert, 1992]. By using a meta-learner, this method tries to induce which classifiers are reliable and which are not. Stacking is usually employed to combine models built by different inducers. The idea is to create a meta-dataset containing a tuple for each tuple in the original dataset. However, instead of using the original input attributes, it uses the predicted classifications by the classifiers as the input attributes. The target attribute remains as in the original training set. A test instance is

first classified by each of the base classifiers. These classifications are fed into a meta-level training set from which a meta-classifier is produced.

This classifier (Meta-classifier) combines the different predictions into a final one. It is recommended that the original dataset should be partitioned into two subsets. The first subset is reserved to form the meta-dataset and the second subset is used to build the base-level classifiers. Consequently the meta-classifier predications reflect the true performance of base-level learning algorithms. Stacking performance can be improved by using output probabilities for every class label from the base-level classifiers. It has been shown that with stacking the ensemble performs (at best) comparably to selecting the best classifier from the ensemble by cross validation [Dzeroski and Zenko, 2004].

*Troika* [Menahem, 2008] was designed to address Stacking problems, namely, the poor performs on multi-class problems [Seewald, 2003]. Troika's ensemble scheme is general purpose which may be used to combine any type of classifiers which were trained on any subgroup of possible classes of a problem's domain. In other words, it is possible with Troika to combine models (classifiers) that were trained on, and therefore may later predict, non congruent datasets, in terms of instances classes. Troika uses three layers of combining classifiers (Figure 1), rather than one, as with Stacking. The result is a better ensemble scheme, usually with multi-class problems where there are enough instances [Menahem, 2008]. In [Menahem, 2008] the writer shows that Troika, in average, is better than using the best classifier selected using cross-validation.

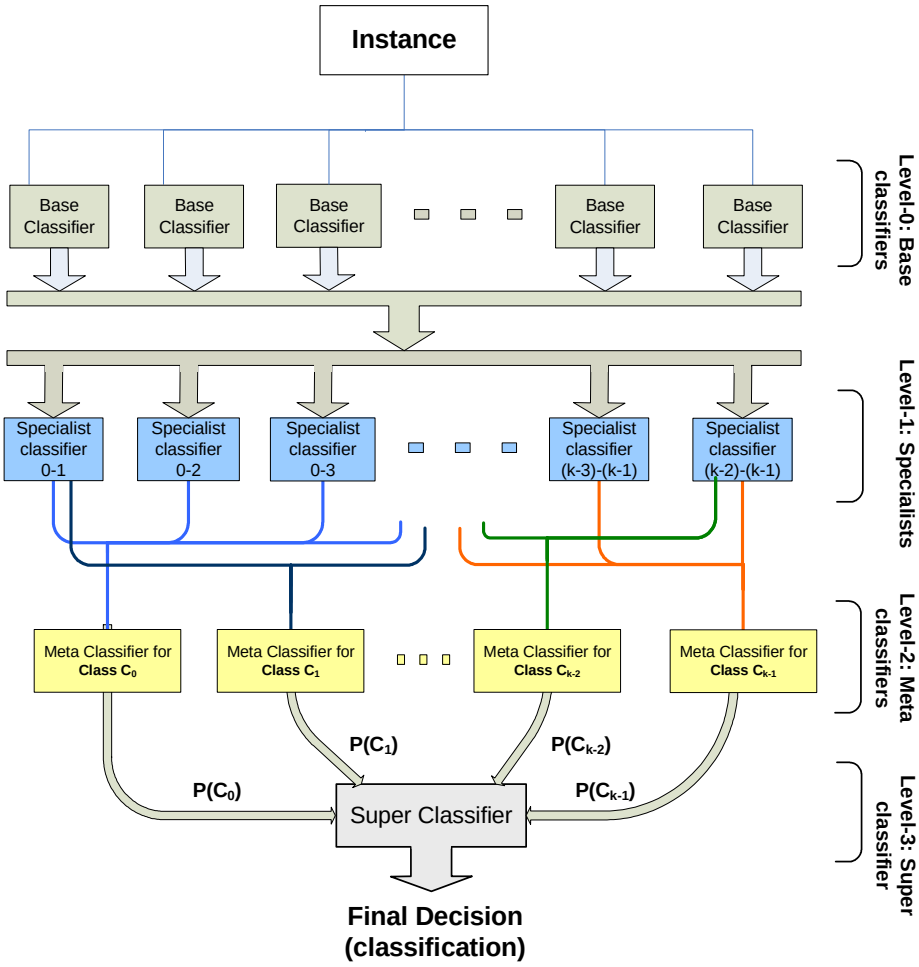


**Fig. 1.** Troika Architecture reveals three layers of combining classifiers; the Specialists classifiers, combine the outputs of the base-classifiers, Meta classifiers combine the Specialists classifiers outputs, and finally, the Super-classifier, combine all Meta-classifiers outputs. The Super classifiers is responsible of output Troika's prediction.

## 3 Evaluation Description

In this section we specify the conditions in which all combination methods had been tested. Our goal was to create a ground on which all combination methods could be correctly compared. First we indicate the algorithms compared. Then we describe the training process and the datasets we had used. Next, we show the metrics we used to

measure the performance of the ensemble schemes, and finally, we display and review the results of the experiments in details.

### 3.1 Base Classifiers and Combination Methods

In the experiment we had used five different inducers as our base classifiers: C4.5 [Quinlan, 1993], KNN [Kibler, 1991], VFI [Guvenir, 1997], OneR [Holte, 1993] and Naïve-Bayes [John and Langley, 1995]. Each inducer belongs to different family of classifiers. For example, C4.5 related to Decision Trees, KNN belong to Lazy classifiers, OneR to Rules, Naïve-Bayes to Bayes classifiers and VFI to general. We have chosen those five inducers because (1) they are from different classifiers family, therefore, may yield different models that eventually will classify differently on some inputs. In an ensemble, the combination of the output of several classifiers is only useful if they disagree about some inputs [Tumer and Ghosh, 1996], and (2) there is a difference in classification accuracy among the inducers; preliminary experiments on our datasets show that C4.5 and KNN are relatively accurate (have better than 80% accuracy) while OneR and Naïve-Bayes are not. This property may differentiate between good ensemble and a bad one, because we expect good ensemble will mostly use the information made by the good induces, while ignoring most of the poor inducers.

In this study, we compared the following combination methods: best classifier (classifier which out-performs all the others, selected using cross validation), Majority Voting, Performance Weighting, Distribution Summation, Bayesian Combination, Naive Bayes, Stacking and Troika. The examined ensemble schemes were implemented in WEKA (Witten and Frank, 2005) in JAVA programming language.

### 3.2 Datasets

We collected a large set of 22,735 benign files and 7690 malware that were identified as Win32 executable and DLLs. For each file three types of features were extracted: n-grams, Portable Executable (PE) features and Function-based features as described bellow.

We developed a tool that extracts *n*-grams from the binary representation of a file. The tool parsed the files using a sliding window producing *n*-grams of varying lengths (denoted by *n*). We extracted vocabularies of 16,777,216, 1,084,793,035, 1,575,804,954 and 1,936,342,220, for 3-gram, 4-gram, 5-gram and 6-gram respectively. Next, each n-gram term was represented using its Term Frequency (TF) and its Inverse Document Frequency (TF-IDF). The TF measure is the number of its appearances in the file, divided by the term with the maximal appearances. IDF is the log value of the number of files in the all repository, divided by the number of files that includes the term. The TF-IDF measure is obtained by multiplying TF by IDF.

In ML applications, the large number of features in many domains presents a huge problem in that some features might not contribute (possibly even harm) the classification accuracy. Moreover, in our case, trimming the number of features is crucial; however, this should be done without diminishing classification accuracy.

The reason for this constraint is that, as shown above, the vocabulary size may exceed billions of features, far more than the processing capacity of any abundant feature selection tool. Moreover, it is important to identify those terms that appear most frequently in files in order to avoid vectors that contain many zeros. Thus, we first extracted the top features based on the Document Frequency (DF) measure, which is the number of files the term was found in. We selected the top 0 - 5,500 features which appear in most of the files, (those with highest DF scores) and the top 1,000 - 6,500 features. Finally, we used three methods of feature selection: Document Frequency, Fisher Score [Golub et al., 1999], and Gain Ratio [Mitchell, 1997] in order to choose the top 50, top 100, top 200 and top 300 *n*-grams, out of the 5,500 *n*-grams, to be used for the evaluation.

After performing rigorous experiments in which we evaluated several inducers on the 192 different n-gram datasets, we chose the top 300 5-grams with TF-IDF values, top 300 6-grams with TF-IDF and top 300 6-grams, with TF datasets providing the best results.

Certain parts of EXE files might indicate that a file is affected by a virus. This may be an inconsistency between different parts of version numbers or internal/external name of a file; some patterns in imported DLLs may be typical for a virus. To verify our hypothesis, we statically extracted different *Portable Executable* (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL). For this purpose, the tool named PE Feature Extractor was implemented in C++. Feature Extractor parses an EXE\DLL file according to PE Format rules. The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE Header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type, file size)
- Optional PE Header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size, debug flags)
- Import Section: which DLLs were imported and which functions from which imported DLLs were used
- Exports Section: which functions were exported (if the file being examined is a DLL)
- Resource Directory: resources used by a given file (e.g., dialogs, cursors)
- Version Information (e.g., internal and external name of a file, version number)

A new method for extracting features from the binary code of the files was implemented and extracted for the experiment. In this method (hereby Function Detector method) we use a J48 decision tree classifier in order to mark beginnings and endings of functions resides in the binary representation of the files. Using those marks we extract function from each file and generate the following attributes (total of 17 attributes):

- Size of file
- File's entropy value
- Total number of detected functions
- Size of the longest function that was detected
- Size of the shortest function that was detected
- Average size of detected functions
- Standard deviation of the size of detected functions

- Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
- Functions ratio- the proportion of number of detected functions from the overall size of the file
- Code ratio- the proportion of the size (in bytes) of detected functions from the overall size of the file

Unfortunately, due to processing power and memory limitations we could not run our ensemble experiments on the entire datasets and we had to reduce the datasets in order to complete the experiments in reasonable time.
Therefore, for the evaluation process we had produced a new dataset from the original dataset by choosing randomly 33% of the instances (i.e., files) in the original datasets and we applied the Gain Ratio feature selection method [Mitchell, 1997] on the *n*-gram based and PE datasets.

As a result, each ensemble scheme in our experiment was tested upon 10 datasets. We had 5 different datasets, and each dataset had two versions. First version contained binary classification (each instance has a 'benign' or 'malicious' labels) and the second, which contained the same instances and attributes, had 8 classes since we had split up the class 'malicious' into 7 sub-classes: 'Virus', 'Worm', ' Flooder', 'Trojan', 'Backdoor', 'Constructor' and 'Nuker'. The instances that got the 'Benign' label in the first version had the same labels in the second version.

**Table 1.** Properties of the data-sets we used in our experiment.

| Dataset | #Classes | #Instances | #Attributes |
|---|---|---|---|
| 6-grams TF-2C | 2 | 9914 | 31 |
| 6-grams TFIDF-2C | 2 | 9914 | 31 |
| 5-grams TFIDF-2C | 2 | 9914 | 31 |
| PE-2C | 2 | 8247 | 30 |
| FD-2C | 2 | 9914 | 17 |
| 6-grmas TF-8C | 8 | 9914 | 31 |
| 6-grams TFIDF-8C | 8 | 9914 | 31 |
| 5-grams TFIDF-8C | 8 | 9914 | 31 |
| PE-8C | 2 | 8247 | 30 |
| FD-8C | 2 | 9914 | 17 |

### 3.3 The Training Process

The training process includes the training of the base-classifiers (the classifiers we later combine) and may include the training of the meta-data set if required. Figure 2 shows the dataset partitioning for each ensemble method. Troika and Stacking training sets have more partitions due to their inner k-fold cross-validation training process. Troika and Stacking train their base classifiers using same percentage of dataset instances, which is 80% of the original train-set (which is 72% of the entire dataset) while the rest 20% of the training instances are reserved for the training of the combining classifier(s). Weighting methods, like voting and Distribution summation, on the other hand, use 100% of the train-set (which is 90% of the original dataset) for the training of the base-classifiers.

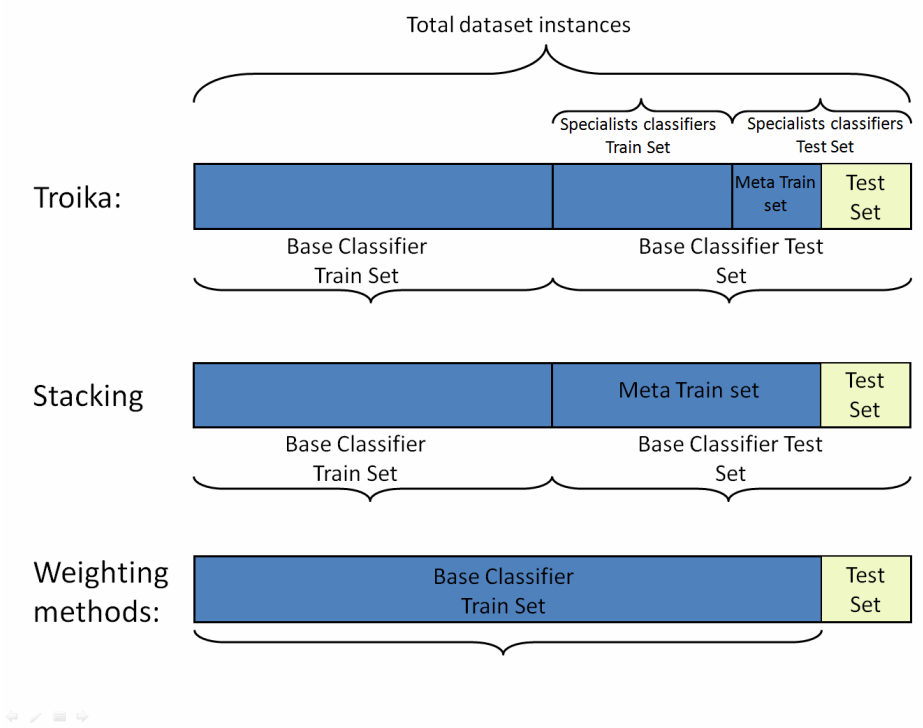


**Fig. 2.** Cross validation scheme for training and testing ensembles methods.

In order to estimate the generalized accuracy, a 10-fold cross-validation procedure was repeated 5 times. In each of the cross-validation iteration, the training set was randomly partitioned into 10 disjoint instance subsets. Each subset was utilized once in a test set and nine times in a training set. The same cross-validation folds were implemented for all algorithms.

### 3.4 Measured Metrics

In order to evaluate the performance of individual classifiers and ensemble combination methods, we used the following common metrics: Accuracy, Area Under the ROC curve, and Training time.

*Accuracy* is the rate of correct (incorrect) predictions made by a model over a data set. Since the mean accuracy is a random variable, the confidence interval was estimated by using the normal approximation of the binomial distribution. Furthermore, the one-tailed paired *t*-test with a confidence level of 95% verified whether the differences in accuracy between a tested ensemble pair were statistically significant. In order to conclude which ensemble performs best over multiple datasets, we followed the procedure proposed in [Demsar, 2006]. In the case of multiple ensemble of classifiers we first used the adjusted Friedman test in order to reject the

null hypothesis and then the Bonferroni–Dunn test to examine whether a specific ensemble performs significantly better than other ensembles.

The *ROC* (Receiver Operating Characteristic) curve is a graph produced by plotting the fraction of true positives (TPR = True Positive Rate) versus the fraction of false positives (FPR = False Positive Rate) for a binary classifier as its discrimination threshold varies. The best possible classifier would thus yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives. We will use the *Area Under the ROC Curve* (AUC) measure in the evaluation process. The AUC value of the best possible classifier will be equal to 1, which means that we can find a discrimination threshold under which the classifier will obtain 0% false positives and 100% true positives. A higher AUC value means that the ROC graph is closer to the optimal threshold. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making. Widely used in medicine, radiology, psychology and other areas for many decades, it has been introduced relatively recently in other areas such as machine learning and data mining.

The *Execution time* (Training time) measure is an applicative one. It has two significances; first, and most logical, heavy time consumption is bad. We would prefer a fast learning ensemble that will yield the best accuracy or area under ROC. Second, the longer time the training of an ensemble takes, the more CPU time it requires, and thus, the more energy it consumes. This is very important on mobile platforms that may be using an ensemble for various reasons.

## 4 Evaluation Results and Analysis

Tables 2-4, present the results obtained using 9 inducers. The 10-fold cross-validation procedure which was repeated five times.

**Table 2.** Comparing the Accuracies of the ensemble algorithms using C4.5, OneR, VFI, KNN and Naïve-Bayes inducers.

| Dataset | Voting | Distribution Summation | Naïve-Bayes Combination | Bayesian Combination | Performance Weighting | Stacking | Troika | Best B.C. | B.C. Name |
|---|---|---|---|---|---|---|---|---|---|
| FD-8C | 80.13 ± 0.88 | 81.53 ± 0.93 | 12.42 ± 1.45 | 81.71 ± 0.75 | 39.72 ± 3.73 | 81.27 ± 1.16 | 82.20 ± 0.81 | 79.87 ± 0.85 | C4.5 |
| FD-2C | 83.12 ± 0.81 | 84.43 ± 0.81 | 78.60 ± 1.25 | 84.55 ± 0.96 | 81.33 ± 0.87 | 85.83 ± 1.09 | 83.70 ± 0.96 | 84.63 ± 1.09 | C4.5 |
| PE-8C | 89.96 ± 1.44 | 90.44 ± 1.52 | 79.39 ± 1.66 | 90.51 ± 1.56 | 88.58 ± 1.73 | 90.87 ± 1.52 | 91.56 ± 1.38 | 91.82 ± 1.46 | KNN |
| PE-2C | 93.89 ± 0.82 | 94.01 ± 0.73 | 94.82 ± 0.75 | 94.83 ± 0.67 | 87.79 ± 1.53 | 95.99 ± 0.76 | 96.30 ± 0.61 | 96.10 ± 0.59 | KNN |
| 6gram tf-8C | 81.15 ± 0.93 | 83.35 ± 0.88 | 42.17 ± 1.46 | 83.13 ± 0.82 | 70.64 ± 1.93 | 83.08 ± 1.02 | 83.42 ± 1.03 | 83.48 ± 1.03 | KNN |
| 6gram tf-2C | 91.57 ± 0.93 | 91.19 ± 0.93 | 67.12 ± 1.57 | 91.39 ± 0.87 | 67.57 ± 1.67 | 93.26 ± 0.81 | 93.31 ± 0.76 | 93.26 ± 0.77 | KNN |
| 5gram tfidf-8C | 81.39 ± 0.89 | 83.90 ± 0.86 | 36.09 ± 1.55 | 83.81 ± 0.80 | 45.23 ± 3.65 | 83.40 ± 0.97 | 84.16 ± 0.96 | 84.12 ± 0.97 | KNN |
| 5gram tfidf-2C | 94.00 ± 0.78 | 92.13 ± 0.83 | 64.28 ± 1.36 | 93.96 ± 0.88 | 64.43 ± 1.36 | 94.03 ± 0.86 | 94.10 ± 0.78 | 94.08 ± 0.82 | KNN |
| 6gram tfidf-8C | 80.48 ± 0.96 | 83.29 ± 0.89 | 59.86± 1.38 | 83.11 ± 0.91 | 46.64 ± 4.47 | 82.27 ± 1.09 | 83.08 ± 0.97 | 83.00 ± 0.98 | KNN |
| 6gram tfidf-2C | 94.24 ± 0.69 | 92.73 ± 0.83 | 53.47 ± 1.97 | 94.63 ± 0.65 | 60.54 ± 1.93 | 94.53 ± 0.65 | 94.55 ± 0.66 | 94.53 ± 0.66 | KNN |
| **Average** | **87.76 ± 0.92** | **88.38 ± 0.92** | **53.47 ± 1.3** | **88.16 ± 0.89** | **65.25 ± 2.29** | **89.25 ± 0.97** | **89.35 ± 0.9** | **88.49 ± 0.92** | |

**Table 3.** Comparing the Area Under the ROC curve (AUC) of the ensemble algorithms using C4.5, OneR, VFI, KNN and Naïve-Bayes inducers.

| Dataset | Voting | Distribution Summation | Naïve-Bayes Combination | Bayesian Combination | Performance Weighting | Stacking | Troika | Best B.C. | B.C. Name |
|---|---|---|---|---|---|---|---|---|---|
| FD-8C | 0.83 ± 0.01 | 0.85 ± 0.02 | 0.55 ± 0.01 | 0.85 ± 0.02 | 0.79 ± 0.02 | 0.86 ± 0.01 | 0.86 ± 0.01 | 0.79 ± 0.02 | C4.5 |
| FD-2C | 0.82 ± 0.01 | 0.84 ± 0.02 | 0.76 ± 0.02 | 0.83 ± 0.02 | 0.81 ± 0.02 | 0.86 ± 0.02 | 0.86 ± 0.02 | 0.8 ± 0.02 | C4.5 |
| PE-8C | 0.99 ± 0.01 | 0.99 ± 0.01 | 0.95 ± 0.02 | 0.99 ± 0.01 | 0.99 ± 0.01 | 0.98 ± 0.01 | 0.99 ± 0.01 | 0.95 ± 0.01 | C4.5 |
| PE-2C | 0.98 ± 0.01 | 0.98 ± 0.00 | 0.94 ± 0.01 | 0.98 ± 0.00 | 0.96 ± 0.01 | 0.99 ± 0.01 | 0.99 ± 0.00 | 0.98 ± 0.0 | C4.5 |
| 6gram tf-8C | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.77 ± 0.01 | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.94 ± 0.01 | 0.95 ± 0.02 | 0.95 ± 0.01 | KNN |
| 6gram tf-2C | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.77 ± 0.01 | 0.97 ± 0.01 | 0.95 ± 0.01 | 0.96 ± 0.01 | 0.97 ± 0.01 | 0.96 ± 0.01 | C4.5 |
| 5gram tfidf-8C | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.73 ± 0.01 | 0.97 ± 0.01 | 0.96 ± 0.01 | 0.94 ± 0.01 | 0.96 ± 0.01 | 0.96 ± 0.01 | C4.5 |
| 5gram tfidf-2C | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.75 ± 0.01 | 0.97 ± 0.01 | 0.95 ± 0.01 | 0.96 ± 0.02 | 0.98 ± 0.01 | 0.95 ± 0.01 | C4.5 |
| 6gram tfidf-8C | 0.96 ± 0.01 | 0.97 ± 0.01 | 0.50 ± 0.00 | 0.97 ± 0.01 | 0.95 ± 0.01 | 0.95 ± 0.01 | 0.95 ± 0.02 | 0.94 ± 0.01 | C4.5 |
| 6gram tfidf-2C | 0.97 ± 0.01 | 0.97 ± 0.01 | 0.80 ± 0.02 | 0.97 ± 0.01 | 0.94 ± 0.01 | 0.97 ± 0.01 | 0.98 ± 0.01 | 0.95 ± 0.01 | C4.5 |
| **Average** | **0.95 ± 0.01** | **0.96 ± 0.01** | **0.75 ± 0.01** | **0.95 ± 0.01** | **0.93 ± 0.01** | **0.95 ± 0.01** | **0.96 ± 0.01** | **0.92 ± 0.01** | |

**Table 4.** Comparing the execution (training) time of the ensemble algorithms using C4.5, OneR, VFI, KNN and Naïve-Bayes inducers.

| Dataset | Voting | Distribution Summation | Naïve-Bayes Combination | Bayesian Combination | Performance Weighting | Stacking | Troika |
|---|---|---|---|---|---|---|---|
| FD-8C | 8.3 ± 0.2 | 8.3 ± 0.2 | 2.1 ± 0.1 | 24.7 ± 0.2 | 24.6 ± 0.3 | 278.6 ± 1.8 | 4693.1 ± 96.4 |
| FD-2C | 0.4 ± 0.0 | 0.4 ± 0.1 | 1.4 ± 0.1 | 21.8 ± 0.8 | 22.1 ± 1.7 | 285.4 ± 48.4 | 170.6 ± 1.5 |
| PE-8C | 5.6 ± 0.7 | 5.7 ± 0.7 | 1.2 ± 0.1 | 10.6 ± 0.3 | 10.8 ± 0.4 | 273.3 ± 8.6 | 1676.7 ± 94.1 |
| PE-2C | 0.9 ± 0.1 | 0.9 ± 0.1 | 0.2 ± 0.0 | 18.8 ± 0.2 | 18.7 ± 0.4 | 160.7 ± 3.8 | 403.9 ± 4.0 |
| 6gram tf-8C | 0.9 ± 0.2 | 0.9 ± 0.2 | 2.3 ± 0.1 | 26.7 ± 0.8 | 27.0 ± 1.1 | 179.3 ± 3.3 | 5136.6 ± 165.0 |
| 6gram tf-2C | 0.6 ± 0.1 | 0.7 ± 0.1 | 1.3 ± 0.1 | 23.3 ± 1.0 | 23.4 ± 2.2 | 162.9 ± 3.2 | 353.9 ± 12.0 |
| 5gram tfidf-8C | 0.9 ± 0.1 | 0.8 ± 0.1 | 2.4 ± 0.1 | 31.3 ± 0.4 | 31.3 ± 0.7 | 190.8 ± 16.5 | 5805.2 ± 141.2 |
| 5gram tfidf-2C | 0.7 ± 0.1 | 0.7 ± 0.1 | 1.5 ± 0.1 | 25.3 ± 0.4 | 25.3 ± 0.7 | 175.2 ± 1.7 | 409.9 ± 5.3 |
| 6gram tfidf-8C | 0.9 ± 0.2 | 0.9 ± 0.1 | 3.2 ± 0.1 | 35.8 ± 1.0 | 37.2 ± 1.2 | 183.5 ± 3.2 | 19267.7 ± 331.8 |
| 6gram tfidf-2C | 0.6 ± 0.1 | 0.6 ± 0.1 | 1.6 ± 0.1 | 25.0 ± 0.3 | 26.0 ± 0.6 | 162.3 ± 3.1 | 388.4 ± 2.2 |
| **Average** | **2.0 ± 0.2** | **2.0 ± 0.2** | **1.7 ± 0.1** | **24.3 ± 0.5** | **24.6 ± 0.9** | **205.2 ± 9.4** | **3830.6 ± 85.3** |

### 4.1 Results Analysis - Accuracy

As can be seen from Table 2, there are some variability among the ensembles mean predictive accuracy. We can see that two ensembles (Troika and Stacking) excels best base-classifier by cross-validation, two other ensembles (Distribution summation and Bayesian combination) are close to best base-classifier from below and the rest three (Voting, Naïve-Bayes and Performance weighting) got the lowest results, away below best base-classifier. A question arises: does this accuracy variability is statistically significant? If the answer is positive, then we would like to know which of the ensembles is better than the others.

We had used the adjusted non-parametric Friedman test in order to check the first hypothesis, that all combination methods perform the same, and the results was that the null-hypothesis could be rejected with a confidence level of 95%. The next step, than, is to decide which of the ensembles, if any, performs best. To that purpose we used the Bonferroni-Dunn test on each pair of ensembles. Below, in Table 5 we show the results of the pairs test results.

**Table 5.** Showing the significance of the difference of the ensembles' accuracies. The "▲" symbol indicates that the degree of accuracy of the row's ensemble scheme was significantly higher than the Column's ensemble scheme at a confidence level of 95%. For example, as can be seen in the table below, Troika's accuracy is higher than Performance-Weighting. The "◄" symbol, on the contrary, indicates the accuracy of the row's ensemble scheme was significantly lower and the "◇" symbol indicates no significant difference.

| Ensemble Method | Stacking | Performance Weighting | Bayesian Combination | Naïve-Bayes Combination | Distribution Summation | Voting | Best Base-classifier |
|---|---|---|---|---|---|---|---|
| **Troika** | ◇ | ▲ | ◇ | ▲ | ◇ | ▲ | ◇ |
| **Stacking** | | ▲ | ◇ | ▲ | ◇ | ◇ | ◇ |
| **Performance Weighting** | | | ◄ | ◇ | ◄ | ◇ | ◄ |
| **Bayesian Combination** | | | | ◇ | ◇ | ▲ | ◇ |
| **Naïve-Bayes Combination** | | | | | ◇ | ◄ | ◄ |
| **Distribution summation** | | | | | | ◇ | ◇ |
| **Voting** | | | | | | | ◄ |

In order to identify the ensemble method that predicts with best accuracy we summarized the result of Table 5 in yet another table, Table 6. This time we calculated a score to each ensemble scheme:

$$score = \#better \big/ \#worse + \varepsilon \qquad \textbf{(8)}$$

where #better is the number of cases where the ensemble was better than other ensemble and #worse is the number of cases where the ensemble was worse than other ensemble.
Equation (8) ensures that the ensemble with maximum #better and minimum #worse will get highest score.

**Table 6.** Score of each ensemble scheme ordered from most successful at top, to worse at bottom. For the score calculation we added a small number (ε=0.001) to the denominator in order to avoid cases of division by zero.

| Ensemble Method | #Worse | #Better | Score | Rank |
|---|---|---|---|---|
| Troika | 0 | 3 | 3001 | 1 |
| Best base-classifier | 0 | 3 | 3001 | 1 |
| Stacking | 0 | 2 | 2001 | 2 |
| Bayesian Combination | 0 | 2 | 2001 | 2 |
| Distribution Summation | 1 | 2 | 1.999 | 3 |
| Voting | 2 | 0 | 0.0005 | 4 |
| Performance Weighting | 5 | 0 | 0.0002 | 5 |
| Naive-Bayes combination | 5 | 0 | 0.0002 | 5 |

### 4.2 Results Analysis - AUC

Table 3 shows the AUC results of all examined ensembles. We can see that all ensembles except Naïve-Bayes excel Best base-classifier by cross-validation. Two ensembles (Distribution summation and Troika) got the best result, which seems significantly better than best-classifier's. All other ensembles AUC results are in the

area of 93% to 95%. Again, we would like to check if the change in ensembles' AUC is significant and if it is, then we would like to know who the best ensemble is.

We had used the adjusted non-parametric Friedman test in order to check the first hypothesis, that all ensemble AUCs are the same, and the results was that the null-hypothesis could be rejected with a confidence level of 95%. Next, we used the Bonferroni-Dunn test on each pair of ensembles to check which ensemble AUC significantly is better. Below, in Table 7 we show the results of this pairs test results.

**Table 7.** Showing the significance of the difference of the ensembles' AUCs. The "▲" symbol indicates that the degree of AUC of the row's ensemble scheme was significantly higher than the Column's ensemble scheme at a confidence level of 95%. he "◄" symbol, on the contrary, indicates the AUC of the row's ensemble scheme was significantly lower and the "◇" symbol indicates no significant difference.

| **Ensemble Method** | **Stacking** | **Performance Weighting** | **Bayesian Combination** | **Naïve-Bayes Combination** | **Distribution Summation** | **Voting** | **Best Base-classifier** |
|---|---|---|---|---|---|---|---|
| **Troika** | ◇ | ▲ | ◇ | ▲ | ◇ | ◇ | ▲ |
| **Stacking** | | ◇ | ◇ | ▲ | ◇ | ◇ | ◇ |
| **Performance Weighting** | | | ◄ | ◇ | ◄ | ◇ | ◇ |
| **Bayesian Combination** | | | | ▲ | ◇ | ◇ | ▲ |
| **Naïve-Bayes Combination** | | | | | ◄ | ◄ | ◇ |
| **Distribution summation** | | | | | | ◇ | ▲ |
| **Voting** | | | | | | | ◇ |

We had summarized the result of Table 7 in table 8 in order to check who has the best AUC. We calculated a score of each ensemble scheme using equation (8).

**Table 8.** Score of each ensemble scheme ordered from most successful at top, to worse at bottom. For the score calculation we added a small number ($\varepsilon$=0.001) to the denominator in order to avoid cases of division by zero.

| **Ensemble Method** | **#Worse** | **#Better** | **Score** | **Rank** |
|---|---|---|---|---|
| Troika | 0 | 3 | 3001 | 1 |
| Bayesian Combination | 0 | 3 | 3001 | 1 |
| Distribution Summation | 0 | 3 | 3001 | 1 |
| Stacking | 0 | 1 | 1001 | 2 |
| Voting | 0 | 1 | 1001 | 2 |
| Best base-classifier | 3 | 1 | 0.3333 | 3 |
| Performance Weighting | 3 | 0 | 0.0003 | 4 |
| Naive-Bayes combination | 5 | 0 | 0.0001 | 5 |

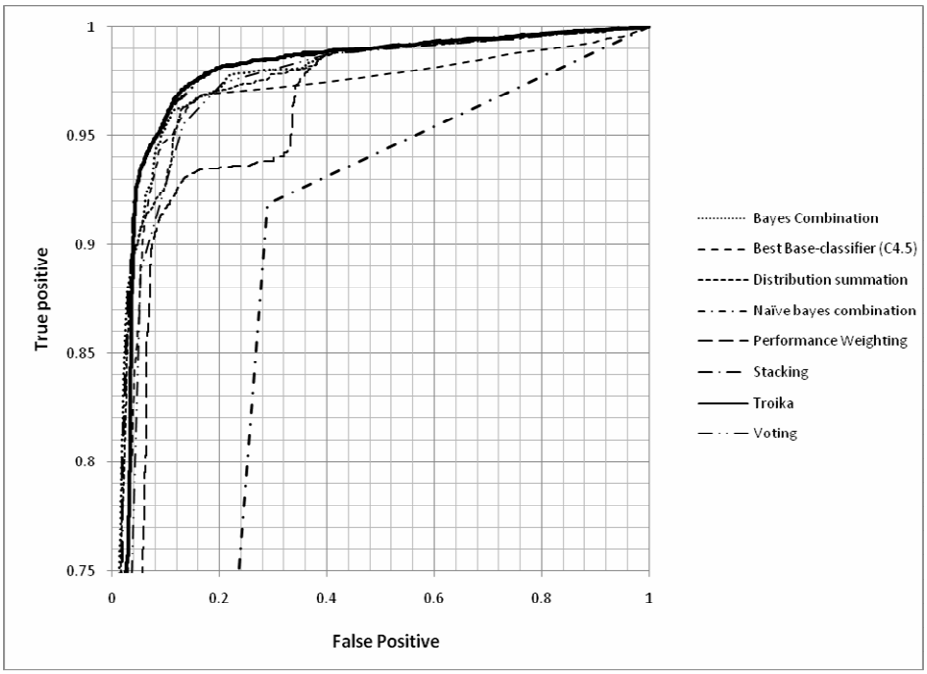


**Fig. 3.** ROC graphs of all eight competitive ensemble combination methods. In spite of the load in the graph above, it still can be easily seen that Troika, in the strong line, excels its opponents when false positive is higher than 0.1. For producing this graph, we tested all ensemble combination methods on "5-gram TFIDF-2C".

### 4.3 Results Analysis – Execution Time

Table 4 shows the Execution time results of the examined ensembles. We can spot two groups of ensembles. The first group consists of all weighing ensembles (Naive-Bayes combination, Performance Weighting, Voting, Distribution Summation and Bayesian Combination) and the common character of this group is execution time of coupe single seconds in average. Second group composed from the meta-combiner ensembles (Troika and Stacking) in which the common character is execution time (training time) of more than 1000 seconds in average. Meta-combiner ensembles consume more time during their training phase due to the fact that they need to train their combiner classifier, usually in a k-fold cross-validation manner. The difference in the results of the two groups is bold and no statistical examination is required.

### 4.4 Results Analysis – Summary

In Table 9 we summarize the accuracy, AUC and execution time scores of all ensemble combination methods. The ranking is from 1 to 5 where '1' is worse and '5' is best.

**Table 9.** Summarizing the accuracy, AUC and execution time rank assigned to each of the ensemble combination methods.

| Ensemble Method | Accuracy | AUC | Execution Time |
|---|---|---|---|
| Troika | 5 | 5 | 1 |
| Stacking | 4 | 4 | 2 |
| Best base classifier | 4 | 3 | 5 |
| Bayesian Combination | 4 | 5 | 4 |
| Distribution Summation | 3 | 5 | 4 |
| Voting | 2 | 4 | 4 |
| Performance Weighting | 1 | 2 | 4 |
| Naive-Bayes combination | 1 | 1 | 4 |

The table shows that Troika has best accuracy and AUC but suffers from worse execution time. Stacking has a disadvantage of accuracy and AUC, but its execution time is somewhat better than Troika's. Choosing Best classifier using cross-validation is not too bad option; in fact, it is a better option than using several other ensemble combination methods like Naïve-Bayes for example. Its accuracy and AUC are mediocre, but it is the fastest to execute. Bayesian Combination is probably the best of its family of weighting ensembles. Its accuracy is almost comparable to Troika, its AUC is a top class and its execution time is very good. Distribution Summation accuracy is average, but it belong to group of ensembles that has best AUC. Its execution time is same as all weighting methods. Voting, Performance Weighting and Naive-Bayes combination have a medium-low accuracy and AUC. Their execution time is as all other weighting methods.

### 4.5 Results Analysis – Dataset Class Labeling

In this experiment, we had 5 different datasets each had two versions. First version contained two classes ('Benign' and 'Malware') and the second contained eight classes ('Benign', 'Virus', Worm' etc.). To remind, the two versions contained the same instances in same order. The only difference was that the instances labeled 'Malware' in the two-class version had been given a more specific label in the multi-class version of the dataset. The purpose of producing second and multi-class version of each dataset was to check if this conversion will yield better ensemble performance. It is sometimes important in the Security domain to discriminate between files of different malware types. For example, it may be important to the security officer to know that a particular threat his organization had encounter was Worm rather than Trojan, because the implication and countermeasures might be completely different. Another important parameter is the FPR which is very important

to reduce to minimal acceptable value. The meaning of FPR in security domain is the proportion of benign files that were classified as malware (an erroneous classification) to the total classified files.

In this part of the work we had examined the hypothesis that using a multi-class datasets will help minimizing the FPR of the tested ensemble. The idea was that since there are more benign files compare to any specific malware family files, than the base-classifiers will be biased towards the Benign files which eventually will yield a lower False-positive cases, on the account of possibly producing more False-negative. The examined parameter is AUC of class benign, since, to our belief, AUC reflects the successfulness of the classifier (the ensemble, in this case) better than that of Accuracy.

The results in Table 10 show that only four ensembles benefit from the transition to multi-class dataset – Distribution-summation, voting, performance weighting and Troika. While the while each of the first three improvements is less than 2 percent Troika manage to improve in about 8 percents, which is not negligible. On the other hand, there are few ensembles that got worse; Naïve-Bayes Combination, for example, suffers from more than 4 percent decrease in AUC due to this transition.

**Table 10.** AUC of ensemble combination methods for the 'Benign' class. The table is divided into two sections. In first section we present the AUC results of multi-class datasets. In the second section we present AUC results of binary-class datasets. In the last row of the table we summarize the improvement in FPR (False-positive rate) of each ensemble when using multi-class instead of using binary-class datasets. The improvement is specified in percents. Negative improvement indicates that using multi-class dataset had worsened FPR.

| Dataset | Bayesian Combination | Distribution Summation | Stacking | Voting | Naive Bayes Combination | Performance Weighting | Troika | C4.5 |
|---|---|---|---|---|---|---|---|---|
| FD-8C | 0.800±0.028 | 0.850±0.017 | 0.799±0.030 | 0.844±0.01 | 0.825±0.056 | 0.844±0.018 | 0.820±0.022 | 0.687±0.040 |
| PE-8C | 0.990±0.007 | 0.990±0.007 | 0.975±0.008 | 0.984 ±0.009 | 0.798±0.030 | 0.988±0.007 | 0.988±0.007 | 0.959±0.011 |
| 6gram tf-8C | 0.969±0.007 | 0.968±0.007 | 0.966±0.010 | 0.965±0.007 | 0.779±0.021 | 0.965±0.008 | 0.968±0.008 | 0.952±0.010 |
| 5gram tfidf-8C | 0.971±0.002 | 0.970±0.005 | 0.966±0.002 | 0.807±0.057 | 0.941±0.063 | 0.972±0.007 | 0.959±0.008 | 0.972±0.005 |
| 6gram tfidf-8C | 0.968±0.015 | 0.967±0.008 | 0.965±0.011 | 0.961±0.010 | 0.863±0.034 | 0.946±0.029 | 0.970±0.012 | 0.949±0.016 |
| **Average 8C** | **0.940±0.07** | **0.949±0.03** | **0.93±0.03** | **0.912±0.04** | **0.841±0.04** | **0.943±0.03** | **0.941±0.04** | **0.904±0.03** |
| FD-2C | 0.843±0.017 | 0.795±0.025 | 0.837±0.014 | 0.829±0.018 | 0.823±0.074 | 0.809±0.018 | 0.644±0.072 | 0.814±0.017 |
| PE-2C | 0.983±0.005 | 0.983±0.007 | 0.977±0.008 | 0.795±0.064 | 0.942±0.067 | 0.984±0.009 | 0.967±0.012 | 0.983±0.005 |
| 6gram tf-2C | 0.971±0.008 | 0.960±0.009 | 0.969±0.008 | 0.966±0.010 | 0.955±0.055 | 0.963±0.012 | 0.808±0.058 | 0.950±0.010 |
| 5gram tfidf-2C | 0.971±0.012 | 0.971±0.007 | 0.967±0.011 | 0.965±0.007 | 0.829±0.062 | 0.949±0.014 | 0.974±0.007 | 0.941±0.046 |
| 6gram tfidf-2C | 0.967±0.012 | 0.953±0.042 | 0.969±0.016 | 0.959±0.015 | 0.836±0.051 | 0.939±0.016 | 0.972±0.009 | 0.952±0.016 |
| **Average 2C** | **0.947±0.01** | **0.932±0.02** | **0.944±0.01** | **0.903±0.02** | **0.877±0.06** | **0.929±0.01** | **0.873±0.03** | **0.928±0.02** |
| **%improvement using 8C** | **-0.752%** | **1.783%** | **-1.026%** | **1.006%** | **-4.100%** | **1.535%** | **7.779%** | **-2.594%** |

## 5 Conclusions and Future Work

In this paper we had examined seven different combination methods plus one alternative for ensemble (selecting best classifier using cross-validation) in the security domain. The goal was to identify the best combination method. We had used 5 different datasets that were produced using various techniques and we had made two versions of each. The first version was binary-classed and second copy was multi-classed. Three interesting parameters where measured; accuracy, AUC and

execution-time. We find it most interesting that each ensemble had its own characters and no two where alike.

From the results we learn that in terms of accuracy, and AUC, Troika is probably the best choice; it is better than Stacking, the only other ensemble in the group of Meta-classifier ensemble. It is also better than best classifier chosen using cross-validation. The benefit of using Troika is double; first, its AUC and accuracy are very high. Second, when using multi-class dataset, it's AUC getting improved in about 8%, producing a model with lower FPR and a nice capability of identifying the malware's specific class. Bayesian-combination, for instance, being second best in accuracy and AUC, had had a fall of 1% in its AUC performance when tested on multi-class dataset. If Execution time is a big concern, than Bayesian Combination will be the best choice, being only second only to Troika in terms of accuracy and AUC.
We do not recommend using Naïve-Bayes ensemble method.

We can also see that the Naïve-Bayes ensemble method only good property is its fast execution time which is the same as all weighting methods. Its accuracy and AUC are fairly bad compare to all other ensembles. One explanation to its poor performance is the fact that its assumptions are probably not taking place; for example, the assumption of independent base-classifier does not exist in practice.

Future work may include the involvement of more inducers for base-classifiers (we had only used 5, while there are plenty more to choose from), some different methods of training the base classifiers – in this research we had used only the trivial training method while there are many other methods that uses binarization of datasets (for example - One against One or One against All) which may yield better accuracy and AUC performance. In addition, in the hereby described experiment, we evaluated different inducers on the same datasets. In future experiments we would like to evaluate the performance of inducers that are trained on different features.

**Acknowledgments.** This research is supported by Deutsche Telecom AG.

## References

1. Abou-Assaleh, T., Cercone, N., Keselj, V., and Sweidan, R., 2004. N-gram based Detection of New Malicious Code. Proc. of the Annual International Computer Software and Applications Conference (COMPSAC'04).
2. Buntine, W., 1990. A Theory of Learning Classification Rules. Doctoral dissertation. School of computing Science, University of Technology, Sydney.
3. Caia D. M., Gokhaleb M., Theilerc J., 2007. Comparison of Feature Selection and Classification Algorithms in Identifying Malicious Executables. Computational Statistics and Data Analysis vol. 51, pp. 3156 – 3172.
4. Clark, P., and Boswell, R., 1991. Rule induction with CN2: Some recent improvements. Proc. of the European Working Session on Learning, pp. 151-163.
5. Demsar, J., 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research, vol. 7, pp. 1-30.
6. Dikinson, J., 2005. The new anti-virus formula. http://www.solunet.com/wp/security/How-to-use-multrd-security.pdf.
7. Dzeroski, S., and Zenko, B., 2004. Is Combining Classifiers with Stacking Better than Selecting the Best One?. Machine Learning, 54(3) 255-273.

8. John, G. H., and Langley, P., 1995. Estimating Continuous Distributions in Bayesian Classifiers. In Proc. of the Conference on Uncertainty in Artificial Intelligence, pp. 338-345.
9. John, G. H., and Langley, P., 1995. Estimating Continuous Distributions in Bayesian Classifiers. Proc. of the Conference on Uncertainty in Artificial Intelligence, San Mateo, pp. 338-345.
10. Golub, T. et al., 1999. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science, vol. 286, pp. 531-537.
11. Guvenir, G. D., 1997. Classification by Voting Feature Intervals. Proc. of the European Conference on Machine Learning, pp. 85-92.
12. Heidari, M., 2004. Malicious Codes in Depth. http://www.securitydocs.com/pdf/2742.pdf.
13. Holte, R., 1993. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. Machine Learning, vol. 11, pp. 63-91.
14. Kabiri, P., and Ghorbani, A. A., 2005. Research on Intrusion Detection and Response: A Survey. International Journal of Network Security, 1(2) 84-102.
15. Kelly, C., Spears, D., Karlsson, C., and Polyakov, P., 2006. An Ensemble of Anomaly Classifiers for Identifying Cyber Attacks. Proc. of the International SIAM Workshop on Feature Selection for Data Mining.
16. Kibler, D. A., 1991. Instance-based learning algorithms. Machine Learning, pp. 37-66.
17. Kienzle, D.M., and Elder, M.C., 2003. Internet WORMS: past, present, and future: Recent worms: a survey and trends. ACM workshop on Rapid malcode (WORM03).
18. Kittler, J., Hatef, M., Duin, R., and Matas, J., 1998. On Combining Classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3) 226-239.
19. Kolter, J., Maloof, M., 2006. Learning to Detect and Classify Malicious Executables in the Wild. Journal of Machine Learning Research, vol. 7, pp. 2721-2744.
20. Kuncheva L.I., 2005. Diversity in Multiple Classifier Systems (Editorial), Information Fusion, 6 (1) 3-4.
21. Mahoney, M.V., 2003 A machine learning approach to detecting attacks by identifying anomalies in network traffic. Ph.D. dissertation, Florida Tech.
22. Menahem, E., 2008. Troika - An Improved Stacking Schema for Classification Tasks. MSc. Thesis in Information System Engineering dep., Ben-Gurion University of the Negev, Israel.
23. Mitchell, T., 1997. Machine Learning. McGraw-Hill.
24. Moskovitch, R., Elovici,Y., and Rokach, L., 2008. Detection of Unknown Computer Worms based on Behavioral Classification of the Host. Computational Statistics and Data Analysis, 52(9) 4544-4566.
25. Mukkamalaa, S., Sunga A.H., and Abrahamb, A., 2005. Intrusion Detection Using an Ensemble of Intelligent Paradigms. Journal of Network and Computer Applications, vol. 28, pp. 167-182.
26. Opitz D. and Shavlik, J., 1996. Generating Accurate and Diverse Members of a Neural Network Ensemble. Advances in Neural Information Processing Systems, vol. 8, pp. 535-541.
27. Quinlan, R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers.
28. Schultz, M., Eskin, E., Zadok, E., and Stolfo, S., 2001. Data Mining Methods for Detection of New Malicious Executables. Proc. of the IEEE Symposium on Security and Privacy, pp. 178-184.
29. Seewald, A. K., 2003. Towards understanding stacking - Studies of a general ensemble learning scheme. PhD Thesis, TU Wien.
30. Tumer, K. and Ghosh, J., 1996. Error Correlation and Error Reduction in Ensemble Classifiers. Connection Science, vol. 8, pp. 385-404.

31. Wang, K., and Stolfo, S.J., 2004. Anomalous Payload-Based Network Intrusion Detection. Proc. of International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 1-12.
32. White, S., 1998. Open problems in computer virus research. IBM Research Center.
33. White, S., 1999. Anatomy of a commercial-grade immune system. IBM Research Center.
34. Wolpert, D. H., 1992. Stacked Generalization. Neural Networks, vol. 5, pp. 241-259.
35. Zhang, B., Yin, J., Hao, J., Zhang D., and Wang, S., 2007. Malicious Codes Detection Based on Ensemble Learning. Autonomic and Trusted Computing, pp. 468-277.

http://web.archive.org/web/20130927072617/http://www.ise.bgu.ac.il/faculty/liorr/Tahan12a.pdf

Journal of Machine Learning Research 1 (2012) 1-48 Submitted 8/11; Published 2/12

# Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features

**Gil Tahan** GILTA@BGU.AC.IL
**Lior Rokach** LIORRK@BGU.AC.IL
**Yuval Shahar** YSHAHAR@BGU.AC.IL
*Department of Information Systems Engineering*
*Ben-Gurion University of the Negev*
*P.O.B. 653, Beer-Sheva, Israel 84105*

**Editor:** Charles Elkan

## Abstract

This paper proposes several novel methods, based on machine learning, to detect malware in executable files without any need for preprocessing, such as unpacking or disassembling. The basic method (Mal-ID) is a new static (form-based) analysis methodology that utilizes common segment analysis in order to detect malware files. By using common segment analysis, Mal-ID is able to discard malware parts that originate from benign code. In addition, Mal-ID utilizes a new kind of feature, termed meta-feature, to better capture the properties of the analyzed segments. Rather than using the entire file, as is usually the case with machine learning based techniques, the new approach detects malware on the segment level. This study also introduces two Mal-ID extensions that improve the Mal-ID basic method in various aspects. We rigorously evaluated Mal-ID and its two extensions with more than ten performance measures, and compared them to the highly rated boosted decision tree method under identical settings. The evaluation demonstrated that Mal-ID and the two Mal-ID extensions outperformed the boosted decision tree method in almost all respects. In addition, the results indicated that by extracting meaningful features, it is sufficient to employ one simple detection rule for classifying executable files.

**Keywords:** Computer Security, Malware Detection, Common Segment Analysis, Supervised Learning

## 1. Introduction

Nowadays the use of the Internet has become an integral part of modern life and Internet browsers are downloading to users a wide variety of content, including new computer software. One consequence of this widespread use is that many computer systems are vulnerable to and infected with malware – malicious software. Malware can be categorized into several groups:

1. Viruses – computer programs that are able to replicate themselves and infect files including the operating systems (OS);

2. Worms – self-replicating computer software that is able to send itself to other computers on a network or the Internet;

©2012 Gil Tahan, Lior Rokach and Yuval Shahar.

**Trend Micro, Inc.**
**EX1019-66**

3. Trojans – a software that appears to perform the desired functionally but is actually implementing other hidden operations such as facilitating unauthorized access to a computer system;

4. Spyware – a software installed on a computer system without the user's knowledge to collect information about the user.

The rate of malware attacks and infections is not yet leveling. In fact, according to O'Farrell (2011) and Symantec Global Internet Security Threat Report Trends for 2010 (Symantec, 2010), attacks against Web browsers and malicious code variants installed by means of these attacks have increased.

There are many ways to mitigate malware infection and spread. Tools such as anti-virus and anti-spyware are able to identify and block or identify malware based on its behavior (Franc and Sonnenburg, 2009) or static features (see Table 1 below). A static feature may be a rule or a signature that uniquely identifies a malware or malware group. While the tools mitigating malware may vary, at their core there must be some classification method to distinguish malware files from benign files.

Warrender et al. (1999) laid the groundwork for using machine learning for intrusions detection. In particular, machine learning methods have been used to analyze binary executables. For example, Wartell el al. (2011) introduce a machine learning-based disassembly algorithm that segments binaries into subsequences of bytes and then classifies each subsequence as code or data. In this paper, the term segment refers to a sequence of bytes of certain size that was extracted from an executable file. While it sequentially scans an executable, it sets a breaking point at each potential code-to-code and code-to-data/data-to-code transition. In addition, in recent years many researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files.

The techniques use three distinct stages:

1. Feature Extraction for file representation – The result of the feature extraction phase is a vector containing the features extracted. An executable content is reduced or transformed into a more manageable form such as:

   (a) Strings – a file is scanned sequentially and all plain-text data is selected.

   (b) Portable Executable File Format Fields – information embedded in Win32 and Win64-bit executables. The information is necessary for the Windows OS loader and application itself. Features extracted from PE executables may include all or part of the following pieces of information: attribute certificate – similar to checksum but more difficult to forge; date/time stamp; file pointer - a position within the file as stored on disk; linker information; CPU type; Portable Executable (PE) logical structure (including section alignment, code size, debug flags); characteristics - flags that indicate attributes of the image file; DLL import section – list of DLLs and functions the executable uses; export section - which functions can be imported by other applications; resource directory –indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of



**Trend Micro, Inc.**
**EX1019-67**

information. e.g., strings for dialogs, images, dialog structures; version information, build information, original filename, etc.); relocation table; and many other features.

(c) n-gram – segments of consecutive bytes from different locations within the executables of length $n$. Each n-gram extracted is considered a feature (Rokach et al., 2008).

(d) Opcode n-gram – Opcode is a CPU specific operational code that performs specific machine instruction. Opcode n-gram refers to the concatenation of Opcodes into segments.

2. Feature Selection (or feature reduction) – During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded. Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed; enhancing generalization capability; improving the interpretability of a model, etc. Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997). Typically, the filter approach is faster than the wrapper approach and is used when many features exist. The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class. The overall expected contribution to the classification is calculated and selection is done according to the highest value. The feature selection measure can be calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006).

3. The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique. Among the many classification techniques, most of which have been implemented in the Weka platform (Witten and Frank, 2005), the following have been used in the context of benign/malware files classification: artificial neural networks (ANNs) (Bishop, 1995) , decision tree (DT) learners (Quinlan, 1993), nave-Bayes (NB) classifiers (John and Langley, 1995), Bayesian networks (BN) (Pearl, 1987), support vector machines (SVMs) (Joachims, 1999), k-nearest neighbor (KNN) (Aha et al., 1991), voting feature intervals (VFI) (Demiröz and Güvenir, 1997), OneR classifier (Holte, 1993), Adaboost (Freund and Schapire, 1999), random forest (Breiman, 2001), and other ensemble methods (Menahem et al., 2009; Rokach, 2010).

To test the effectiveness of ML techniques, in malware detection, the researchers listed in Table 1 conducted experiments combining various feature extraction methods along with several feature selection and classification algorithms.

Ye et al. (2009) suggested using a mixture of features in the malware-detection process. The features are called Interpretable Strings as they include both programs' strings and strings representing the API execution calls used. The assumption is that the strings capture important semantics and can reflect an attacker's intent and goal. The detection process starts with a feature parser that extract the API function calls and looks for a sequence of consecutive bytes that forms the strings used. Strings must be of the same encoding and



**Trend Micro, Inc.**
**EX1019-68**

character set. The feature-parser uses a corpus of natural language to filter and remove non-interpretable strings. Next, the strings are ranked using the Max-Relevance algorithm. Finally, a classification model is constructed from SVM ensemble with bagging.

Ye et al. (2010) presented a variation of the method, presented above, that utilizes Hierarchical Associative Classifier (HAC) to detect malware from a large imbalanced list of applications. The malware in the imbalanced list were the minority class. The HAC methodology also uses API calls as features. Again, the associative classifiers were chosen due to their interpretability and their capability to discover interesting relationships among API calls. The HAC uses two stages: to achieve high recall, in the first stage, high precision rules for benign programs (majority class) and low precision rules for minority class are used, then, in the second stage, the malware files are ranked and precision optimization is performed.

Instead of relying on unpacking methods that may fail, Dai et al. (2009) proposed a malware-detection system, based on a virtual machine, to reveal and capture the needed features. The system constructs classification models using common data mining approaches. First, both malware and benign programs are executed inside the virtual machine and the instruction sequences are collected during runtime. Second, the instruction sequence patterns are abstracted. Each sequence is treated as a feature. Next, a feature selection process in performed. In the last stage a classification model is built. In the evaluation the SVM model performed slightly better then the C4.5 model.

Yu et al. (2011) presented a simple method to detect malware variants. First, a histogram is created by iterating over the suspected file binary code. An additional histogram is created for the base sample (the known malware). Then, measures are calculated to estimate the similarity between the two histograms. Yu et al. (2011) showed that when the similarity is high, there is a high probability that the suspected file is a malware variant.

The experiments definitely proved that is possible to use ML techniques for malware detection. Short n-gram were most commonly used as features and yielded the best results. However, the researchers listed did not use the same file sets and test formats and therefore it is very difficult or impossible to compare the results and to determine what the best method under various conditions is. Table 2 presents predictive performance results from various researches.

When we examined the techniques, several insights emerged:

1. All applications (i.e. software files tested in the studies) that were developed using a higher level development platforms (such as Microsoft Visual Studio, Delphi, Microsoft.Net) contain common code and resources that originate from common code and resource libraries. Since most malware are also made of the same common building blocks, we believe it would be reasonable to discard the parts of a malware that are common to all kinds of software, leaving only the parts that are unique to the malware. Doing so should increase the difference between malware files and benign files and therefore should result in a lower misclassification rate.

2. Long n-gram create huge computational loads due to the number of features. This is known as the curse of dimensionality (Bellman et al., 1966). All surveyed n-gram experiments were conducted with n-gram length of up to 8 bytes (in most cases 3-byte n-gram) despite the fact that short n-gram cannot be unique by themselves. In many



**Trend Micro, Inc.**
**EX1019-69**

| Study | Feature Representation | Feature Selection | Classifiers |
|---|---|---|---|
| Schultz et al. (2001) | PE, Strings, n-gram | NA | RIPPER, Nave Bayes, and Multi-Nave Bayes |
| Kolter and Maloof (2004) | n-gram | NA | TFIDF, Nave Bayes, SVM, Decision Trees, Boosted Decision Trees, Boosted Nave Bayes, and Boosted SVM |
| Abou-Assaleh et al. (2004) | n-gram | NA | K-Nearest Neighbors |
| Kolter and Maloof (2006) | n-gram | Information-Gain | K-Nearest Neighbors, Nave Bayes, SVM, Decision Trees, Boosted Decision Trees, Boosted Nave Bayes, and Boosted SVM. |
| Henchiri and Japkowicz (2006) | n-gram | Hierarchical feature selection | Decision Trees, Nave Bayes, and SVM |
| Zhang et al. (2007) | n-gram | Information-Gain | Probabilistic Neural Network |
| Elovici et al. (2007) | PE and n-gram | Fisher Score | Bayesian Networks, Artificial Neural Networks, and Decision Trees |
| Ye et al. (2008) | PE | Max-Relevance | Classification Based on Association (CBA) |
| Dai et al. (2009) | instruction sequence | Contrast measure | SVM |
| Ye et al. (2009) | PE (API) | Max-Relevance | SVM ensemble with bagging |
| Ye et al. (2010) | PE (API) | Max-Relevance | Hierarchical Associative Classifier (HAC) |
| Yu et al. (2011) | histogram | NA | Nearest Neighbors |

Table 1: Recent research in static analysis malware detection in chronological order.



**Trend Micro, Inc.**
**EX1019-70**

| **Method** | **Study** | **Features** | **Feature selection** | **FPR** | **TPR** | **Acc** | **AUC** |
|---|---|---|---|---|---|---|---|
| Artificial Neural Network | Elovici et al. (2007) | 5grams | Fisher Score top 300 | 0.038 | 0.89 | 0.94 | 0.96 |
| Bayesian Network | Elovici et al. (2007) | 5grams | Fisher Score top 300 | 0.206 | 0.88 | 0.81 | 0.84 |
| Bayesian Network | Elovici et al. (2007) | PE | n/a | 0.058 | 0.93 | 0.94 | 0.96 |
| Decision Tree | Elovici et al. (2007) | 5grams | Fisher Score top 300 | 0.039 | 0.87 | 0.93 | 0.93 |
| Decision Tree | Elovici et al. (2007) | PE | n/a | 0.035 | 0.92 | 0.95 | 0.96 |
| Classification Based on Association | Ye et al. (2008) | PE | Max-Relevance | 0.125 | 0.97 | 0.93 | — |
| Boosted Decision Tree | Kolter and Maloof (2006) | 4grams | Gain Ratio | — | — | — | 0.99 |

Table 2: Comparison of several kinds of machine learning methods. FPR, TPR, ACC and AUC refers to False Positive Rate, True Positive Rate, Accuracy and the Area Under Receiver Operating Characteristic (ROC) Curve as defined in Section 3.2.



**Trend Micro, Inc.**
**EX1019-71**

cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware. As a result, current techniques using short n-gram rely on complex conditions and involve many features for detecting malware files.

The goal of this paper is to develop and evaluate a novel methodology and supporting algorithms for detecting malware files by using common segment analysis. In the proposed methodology we initially detect and nullify, by zero patching, benign segments and therefore resolve the deficiency of analyzing files with segments that may not contribute or even hinder classification. Note that, when a segment represents at least one line of code developed using a high level language; it can address the second deficiency of using short features that may be meaningless when considered alone. Additionally, we suggest utilizing meta-features instead of using plain features such as n-gram. A meta-feature is a feature that captures the essence of plain feature in a more compact form. Using those meta-features, we are able to refer to relatively long sequences (64 bytes), thus avoiding the curse of dimensionality.

## 2. Methods

As explained in Section 1, our basic insight is that almost all modern computer applications are developed using higher level development platforms such as: Microsoft Visual Studio, Embarcadero Delphi, etc. There are a number of implications associated with utilizing these development platforms:

1. Since application development is fast with these platforms, both legitimate developers and hackers tend to use them. This is certainly true for second-stage malware.

2. Applications share the same libraries and resources that originated from the development platform or from third-party software companies. As a result, malware that has been developed with these tools generally resembles benign applications. Malware also tends, to a certain degree, to use the same specialized libraries to achieve a malicious goal (such as attachment to a different process, hide from sight with root kits, etc). Therefore it may be reasonable to assume that there will be resemblances in various types of malware due to sharing common malware library code or even similar specific method to perform malicious action. Of course such malware commonalities cannot be always guaranteed.

3. The size of most application files that are being produced is relatively large. Since many modern malware files are in fact much larger than 1 MB, analysis of the newer applications is much more complex than previously when the applications themselves were smaller as well as the malware attacking them.

The main idea presented in this paper is to use a new static analysis methodology that utilizes common segment analysis in order to detect files containing malware. As noted above, many applications and malware are developed using the development platforms that include large program language libraries. The result is that large portions of executable code originate from the program language libraries. For example, a worm malware that



**Trend Micro, Inc.**
**EX1019-72**

distributes itself via email may contain a benign code for sending emails. Consequently, since the email handling code is not malicious and can be found in many legitimate applications, it might be a good idea to identify code portions that originate from a benign source and disregard them when classifying an executable file. In other words, when given an unclassified file, the first step would be to detect the file segments that originated from the development platform or from a benign third party library (termed here the Common Function) and then disregard those segments. Finally, the remaining segments would be compared to determine their degree of resemblance to a collection of known malwares. If the resemblance measure satisfies a predetermined threshold or rule then the file can be classified as malware.

To implement the suggested approach, two kinds of repositories are defined:

1. **CFL – Common Function Library.** The CFL contains data structures constructed from benign files.

2. **TFL – Threat Function Library**. The TFL contains data structures constructed from malware without segments identified as benign (i.e., segments that appears in benign files).

Figure 1 presents the different stages required to build the needed data structures and to classify an application file. As can be seen in this figure, our Mal-ID methodology uses two distinct stages to accomplish the malware detection task: setup and detection. The setup stage builds the CFL. The detection phase classifies a previously unseen application as either malware or benign. Each stage and each sub-stage is explained in detail in the following subsections. The Mal-ID pseudo code is presented in Figure 2.

### 2.1 The setup phase

The setup phase involves collecting two kinds of files: benign and malware files. The benign files can be gathered, for example, from installed programs, such as programs located under Windows XP program files folders. The malware files can, for example, be downloaded from trusted dedicated Internet sites, or by collaborating with an anti-virus company. In this study the malware collection was obtained from trusted sources. In particular, Ben-Gurion University Computational Center provided us malware that were detected by them over time. Each and every file from the collection is first broken into 3-grams (three consecutive bytes) and then an appropriate repository is constructed from the 3-grams. The CFL repository is constructed from benign files and the TFL repository is constructed from malware files. These repositories are later used to derive the meta-features – as described in Section 2.2.

Note that in the proposed algorithm, we are calculating the distribution of 3-grams within each file and across files, to make sure that a 3-gram belongs to the examined segment and thus associate the segment to either benign (CFL) or malware (TFL). Moreover, 3-grams that seem to appear approximately within the same offset in all malware can be used to characterize the malware. Before calculating the 3-grams, the training files are randomly divided into 64 groups.

The CFL and TFL repositories share the same data structure:



**Trend Micro, Inc.**
**EX1019-73**

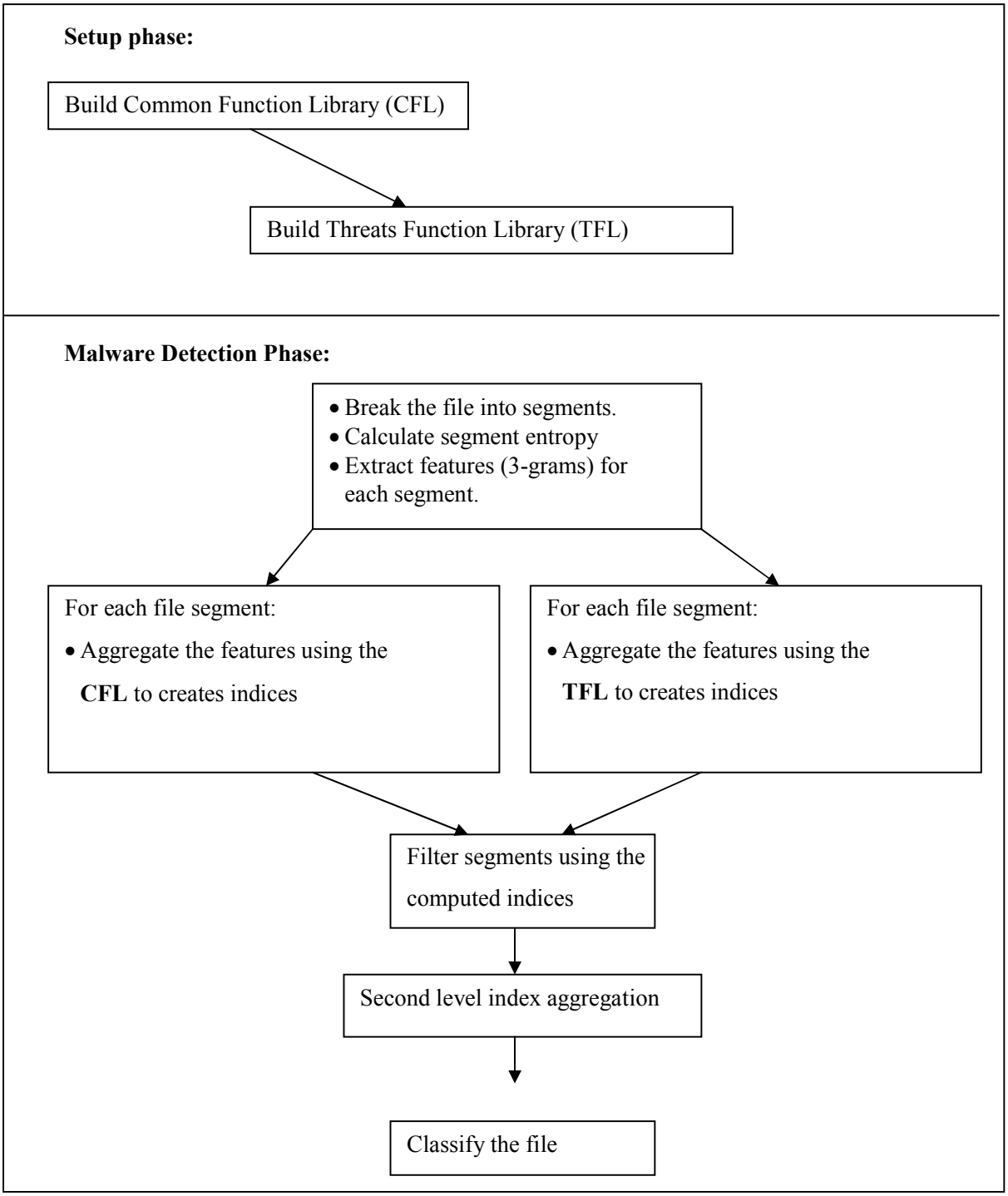


Figure 1: The Mal-ID method for detecting new malware applications.

1. 3-gram-files-association: $2^{24}$ entries, each of 64 bits. A bit value of 1 in a cell (i, j) indicates the appearance of a specific 3-gram i in the $j^{th}$ group of files. The 64-bit entry size was selected since a previous study showed that this size is the most cost effective in terms of detection performance vs. storage complexity (Tahan et al., 2010). Other implementations may use larger entries.

1. 3-gram-relative-position-within-file: $2^{24}$ entries, each of 64 bits. A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the $j^{th}$ internal segment of a file (assuming the file is divided into 64 equal length segments).

The CFL is constructed first and then the TFL:

1. Each file from the malware collection is broken into segments. The Mal-ID implementation has used 64-byte segments.

2. Each segment is broken into 3grams and then tested against the CFL using the algorithm and features described next. Segments that are not in the CFL are added to the TFL.

It is important to note that the end result is the TFL, a repository made of segments found only in malware and not in benign files.

### 2.2 The Detection phase

The Mal-ID basic is a feature extraction process followed by a simple static decision rule.

It operates by analyzing short segments extracted from the file examined. Each segment comprises a number of 3-grams depending on the length of the segment (e.g. a segment of length 4 bytes is comprised from two 3-grams that overlap by two bytes). Three features can be derived for each segment: Spread, MFG, and Entropy. The Spread and the MFG features are derived using the data structures prepared in the setup stage described in Section 2.1 above.

The definition and motivation behind the new features are hereby provided:

1. **Spread**: Recall that in the Mal-ID setup phase each file in the training set has been divided into 64 relative-position-areas. The Spread feature represents the spread of the signature's 3-grams along the various areas for all the files in a given repository. The Spread feature can be calculated as follows: for each 3-gram, first retrieve the 3-gram-relative-position-within-file bit-field, and then perform 'And' operations over all the bit-fields and count the resulting number of bits that are equal to 1. In other words, spread approximates the maximum number of occurrences of a segment within different relative locations in train sets. For example, a Spread equal to 1 means that the segment appears (at most) in one relative location in all the files.

2. **MFG:** the maximum total number of file-groups that contain the segment. The MFG is calculated using the 3-gram-files-association bit-field, in the same manner that spread is calculated.



**Trend Micro, Inc.**
**EX1019-75**

3. **Entropy:** the entropy measure of the bytes within a specific segment candidate. In addition to the new estimators presented above, the entropy feature is also used to enable to identification of compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.

Note that the features, as described above, are in fact **meta-features** as they are used to represent *features of features* (features of the basic 3-grams). As explained next, using these meta-features, Mal-ID can refer to relatively long sequences (64 bytes), thus avoiding the data mining problem known as "the curse of dimensionality", and other problems caused when using short n-gram as features. The advantages of using Mal-ID meta-features will be demonstrated in the evaluation results section and in the discussion section.

### 2.3 The Mal-ID basic detection algorithm

The input for the Mal-ID method is an unclassified executable file of any size. Once the setup phase has constructed the CFL and the TFL, it is possible to classify a file F as benign or as malware using the algorithm presented in Figure 2.

1. Line 1. Divide file F into S segments of length L. All segments are inserted into a collection and any duplicated segments are removed. The end result is a collection of unique segments. The Mal-ID implementation uses 2000 segments that are 64-bytes in length.

2. Line 3. For each segment in the collection:

   (a) Line 5. Calculate the entropy for the bytes within the segment.

   (b) Line 6. The algorithm gets two parameters EntropyLow and EntropyHigh. The entropy thresholds are set to disregard compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information. In this line we check if the entropy is smaller than EntropyLow threshold or entropy is larger than EntropyHigh. Is so then discard the segment and continue segment iteration. Preliminary evaluation has found the values of EntropyLow=0.5 and EntropyHigh=0.675 maximize the number of irrelevant segments that can be removed.

   (c) Line 9. Extract all 3-grams using 1 byte shifts.

   (d) Line 11. Using the CFL, calculate the CFL-MFG index.

   (e) Line 12. If the CFL-MFG index is larger than zero, then discard the segment and continue segment iteration. The segment is disregarded since it may appear in benign files.

   (f) Line 14. Using the TFL, calculate the TFL-MFG index

   (g) Line 15. The algorithm gets the ThreatThreshold parameter which indicates the minimum occurrences a segment should appear in the TFL in order to be qualified as malware indicator. In this line we check if the TFL-MFG index is smaller or equal to the ThreatThreshold. If so then discard the segment and continue with segment iteration. In the Mal-ID implementation only segments



**Trend Micro, Inc.**
**EX1019-76**

that appear two times or more remain in the segment collection. Obviously a segment that does not appear in any malware cannot be used to indicate that the file is a malware.

(h) Line 17. Using the TFL calculate the TFL-Spread index

(i) Line 18. The algorithm gets the SR parameter which indicates the Spread Range required. If the TFL-Spread index equals zero or if it is larger than what we term SR threshold, then discard the segment and continue segment iteration. The purpose of these conditions is to make sure that all 3-grams are located in at least 1 segment in at least 1 specific relative location. If a segment is present in more than SR relative locations it is less likely to belong to a distinct library function and thus should be discarded. In our Mal-ID implementation, SR was set to 9.

(j) Lines 21-25 (optional stage, aimed to reduce false malware detection). A segment that meets all of the above conditions is tested against the malware file groups that contain all 3-gram segments. As a result, only segments that actually reside in the malware are left in the segment collection. Preliminary evaluation showed that there is no significant performance gain performing this stage more than log (SegmentLen) * NumberOfMalwareInTraining iterations.

3. Lines 28-30. Second level index aggregation - Count all segments that are found in malware and not in the CFL.

4. Line 32. Classify – If there are at least X segments found in the malware train set (TFL) and not in the CFL then the file is malware; otherwise consider the file as benign. We have implemented Mal-Id with X set to 1.

Please note that the features used by Mal-ID algorithm described above are in fact meta-features that describe the 3-grams features. The advantages of using Mal-ID meta-features will be described in the following sections.

#### 2.3.1 Mal-ID complexity

**Proposition 1** *The computational complexity of the algorithm in Figure 2 is* $O(SN + log(SL) \cdot M \cdot MaxMalSize)$ *where* $SN$ *denotes the number of segments;* $SL$ *denotes segment length;* $M$ *denotes the number of malware in the training set; and* $MaxMalSize$ *denotes the maximum length of a malware.*

**Proof** The computational complexity of the algorithm in Figure 2 is computed as follows: the GenerateSegmentCollection complexity is $O(SN)$; the complexity of loop number 1 (lines 3-21) is $O(SN + log(SL) \cdot M \cdot MaxMalSize)$; the complexity of loop number 2 (lines 26-29) is $O(SN)$. Thus, the overall complexity is $O(SN + log(SL) \cdot M \cdot MaxMalSize)$. ■



**Trend Micro, Inc.**
**EX1019-77**

```
SegmentColl=GenerateSegmentCollection(FileContent,SegmentsRequired,SegmentLen);
SegmentCheck=0;
ForEach Segment in SegmentColl do
    {
    Entropy  = Entropy(Segment.string);
    If (Entropy<=EntropyLow) or (Entropy>= EntropyHigh) then
       {SegmentColl.delete(Segment); continue; }

    Segment3Grams:=SegmentTo3Grams(Segment);

    CFL_MFG = CFL.Count_Files_With_All_3gram (Segment3Grams)
    If (CFL_MFG>0) then { SegmentColl.delete(Segment); continue; }

    TFL_MFG = TFL.Count_Files_With_All_3gram (Segment3Grams)
    If (TFL_MFG< ThreatsThreshold) then { SegmentColl.delete(Segment); continue; }

    TFL_spread   = TFL.CalcSpread (Segment3Grams);
    If (TFL_spread =0) or  (TFL_spread >SR)  then
       {SegmentColl.delete(Segment); continue; }

    // optional stage
    SegmentCheck++;
    If (SegmentCheck>log(SegmentLen)*NumberOfMalwareInTraining) then continue;
    InMalwareFile  = TFL.SearchInMalwareFiles(Segment);  //search by bit-fields
    If  not InMalwareFile  then { SegmentColl.delete(Segment); continue; }
    }

SegmentsInMalwareOnly = 0;
ForEach Segment in SegmentColl do
    { SegmentsInMalwareOnly  = SegmentsInMalwareOnly +1; }

Malware_Classfication_Result = SegmentsInMalwareOnly > ThreatSegmentThreshold;
```

Figure 2: Mal-ID pseudo code.

### 2.4 Combining Mal-ID with ML generated models

We attempted to improve the *Mal-ID basic* method by using Mal-ID features with various classifiers, but instead of using the Mal-ID decision model described in Section 2, we let various ML algorithms build the model using the following procedure:

1. We apply the common segment analysis method on the training set and obtain a collection of segments for both the CFL and the TFL as explained in Section 2.

2. For each file's segment, we calculated the CFL-MFG, TFL-MFG and the TFL-spread based on the CFL and TFL. The entropy measure is calculated as well.

3. We discretized the numeric domain of the above features using the supervised procedure of Fayyad and Irani (1993). Thus for each feature we found the most representative sub-domains (bins).

4. For each file we count the number of segments associated with each bin. Each frequency count is represented twice: once as absolute numbers (number of segments) and then as a proportional distribution.

5. An induction algorithm is trained over the training set to generate a classifier.

We compare the following three machine learning induction algorithms:

1. C4.5 - Single Decision Tree

2. RF - Rotation Forest (Rodriguez et al., 2006) using J48 decision tree as base classifier. The algorithm was executed with 100 iterations and the PCA method for projecting the data in every iteration.

3. NN - A multilayer perception with one hidden layer trained over 500 epochs using back-propagation.

Finally, using the model is used to detect the malware among the files in the test set.

### 2.5 Combining Mal-ID with ML models post processing

We have attempted to improve the *Mal-ID basic* method by using the following procedure:

1. First, the *Mal-ID basic* method is used to construct the CFL and TFL. This stage is performed only once before the file classification starts.

2. Next, zero patch each malware in the training set as follows: Iterate over all of the file segments and perform common segment analysis to detect the segments that appear in the CFL. The benign segments (the segments that appear in the CFL) are zero patched in an attempt to reduce the number of n-gram that are clearly not relevant for detecting segments that appear only in malware. The end result is a new file with the same length that has zeros in the benign segments.

3. Finally, construct a classification model using Rotation Forest using J48 decision tree as base classifier. The patched malware collection and the unchanged benign file collection are used for training.

To classify a file we first have to zero-patch the file as explained above then use the classification model created earlier.

## 3. Experimental Evaluation

In order to evaluate the performance of the proposed methods for detecting malwares, a comparative experiment was conducted on benchmark datasets. The proposed methods were compared with the method presented in the research of Kolter and Maloof (2004).



**Trend Micro, Inc.**
**EX1019-79**

The research of Kolter and Maloof (2006) found that the combination of 500 4-grams with gain ratio feature selection and boosted decision tree provides the best performance over many other evaluated method variations. We will refer to our variation of Kolter and Maloof method as *GR500BDT* as it uses Gain Ratio feature selection, **500** 4-grams, and Boosted Decision Tree classifier. The *GR500BDT* method was specifically selected because it was the best method known to us.

The following terms will be used when referring to the various methods:

1. *GR500BDT* – Our baseline method, which is described above.
2. *Mal-IDP+GR500BDT* – As explained in Section 2.5, we use Mal-ID to zero patch common segments in the test files, and then use *GR500BDT* as usual.
3. *Mal-ID basic* – *Mal-ID basic* method as explained in Section 2.
4. *Mal-IDF+*<induction algorithm> – as detailed in Section 2.4, Mal-ID features will be used by induction algorithm.
   (a) *Mal-IDF+RF* - Mal-ID features with Rotation Forest classification
   (b) *Mal-IDF+C4.5* - Mal-ID features with C4.5
   (c) *Mal-IDF+NN* - Mal-ID features with a multilayer perception.

Specifically, the experimental study had the following goals:

1. To examine whether the proposed basic methods, could detect malware while keeping the false alarm rate as small as possible.
2. Compare the performance of the various *Mal-ID basic* extensions.
3. To analyze the effect of the common library size (benign and malware) on performance.

The following subsections describe the experimental set-up and the results that were obtained.

### 3.1 Experimental Process

The main aim of this process was to estimate the generalized detection performance (i.e., the probability that a malware was detected correctly). The files repository was randomly partitioned into training and test sets. The process was repeated 10 times and we report the average result. The same train-test partitioning was used for all algorithms.

For evaluating the proposed methodology 2627 benign files were gathered from programs installed under Windows XP program files folders, with lengths ranging from 1Kb to 24MB. An additional 849 malware files were gathered from the Internet with lengths ranging from 6Kb to 4.25MB (200 executables were above 300KB). The detailed list of examined executables can be obtained in the following URL: http://www.ise.bgu.ac.il/faculty/liorr/List.rar. The malware and benign file sets were used without any decryption, decompression or any other preprocessing. The malware types and frequencies are presented in Figure 3. The evaluation computer used an Intel Q6850 CPU with 4GB of RAM. The processing time was measured using only 1 CPU core, although the implemented algorithm natively supported multiple cores.



**Trend Micro, Inc.**
**EX1019-80**

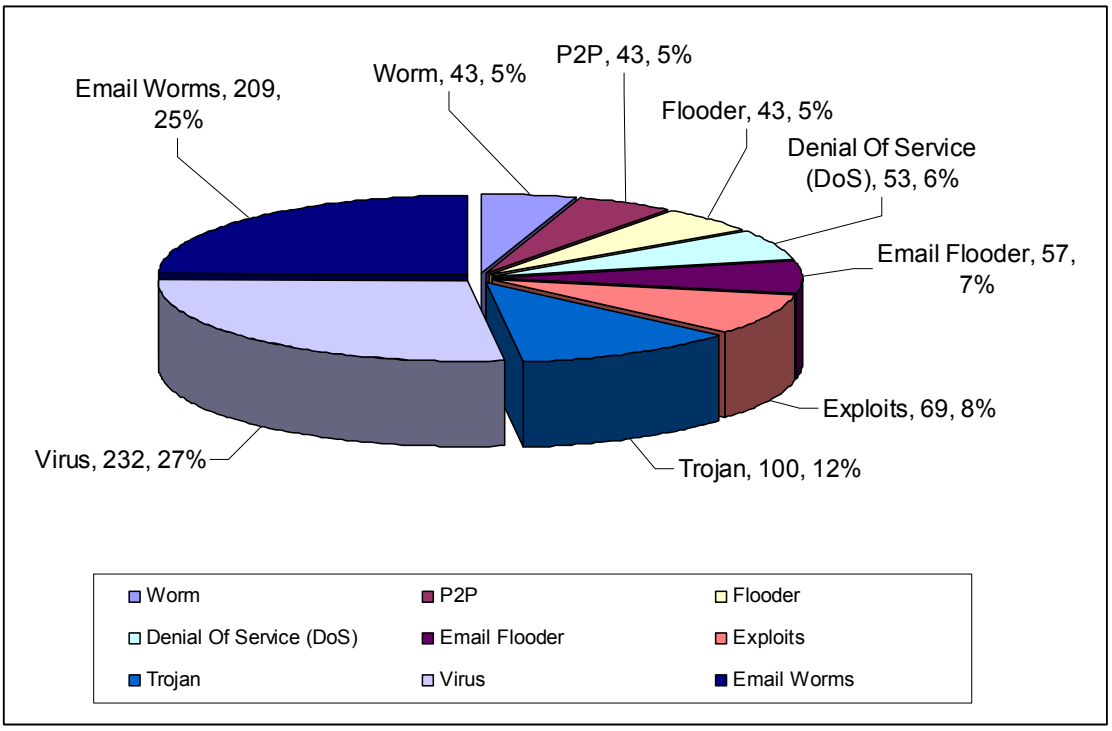


Figure 3: Distribution of malware types in dataset.

### 3.2 Evaluation Measures

We used the following performance measures:

- TP = true positive
- FP = false positive
- TN = true negative
- FN = false negative
- FPR = FP / N = FP / (FP + TN) = false positive rate
- TPR = TP / P = TP / (TP + FN) = true positive rate (also known as sensitivity)
- PPV = TP / (TP + FP) = positive predictive value
- NPV = TN / (TN + FN) = negative predictive value
- ACC = (TP + TN) / (P + N) = accuracy
- BER = 0.5(FN/P + FP/N) = balanced error rate
- BCR = 1- BER = balanced correctness rate



**Trend Micro, Inc.**
**EX1019-81**

- AUC = area under receiver operating characteristic (ROC) curve

Our measures, such as PPV versus NPV, as well as BER or BCR, try to address the important case of an unbalanced positive/negative instance case mix, which is often ignored in the literature. Given the low rate of malware versus benign code, accuracy might be a misleading measure. For example, a "*Maximal Class Probability*" (MPC) classifier is a classifier that always predicts the most frequent class. Thus, an MPC predicting "BENIGN" for every instance in an environment where 99% of the files are benign would, indeed, be 99% accurate. That would also be its NPV, since there is a 99% probability that the MPC is right when it predicts that the file is benign. However, its PPV would be 0, or rather, undefined, since it never predicts a positive class; in other words, its sensitivity to positive examples is 0.

Furthermore, unlike many studies in the information security literature, we use the cross-entropy as one of our major performance measures. The cross-entropy described by Caruana et al. (2004). It is also referred in the literature by the terms *negative log-likelihood* or *log-loss*. Let $p(x_i)$ represents the posterior probability of the instance $x_i$ to be associated with the malware class according to the classifier. The *average cross-entropy* is defined as the average over all $m$ test instances:

$$Entropy = \frac{1}{m}\sum_{i=1}^{m} I(x_i)$$

where the *cross-entropy* for a certain case is defined as:

$$I\left(x_i\right) = \begin{cases} -logP(x_i) & \text{if } x_i \text{ is malware} \\ -log\left(1 - P(x_i)\right) & \text{otherwise} \end{cases}$$

The use of cross-entropy as a measure of knowledge gain allows us to plot the improvement in a learning process, given an increasing number of examples, by noting whether there is a positive information gain (i.e., a reduction in the entropy after learning, compared to the entropy of the previous learning phase). In particular, we would expect an algorithm that really learns something about the classification of both the positive and negative cases to demonstrate a positive monotonic improvement in the cross-entropy measure. It is important to show this positive monotonic improvement since we would prefer an algorithm that generates classifiers in a stable fashion. Such an algorithm can be considered as more trustworthy than an algorithm whose learning curve might be chaotic.

### 3.3 Results

The following sections describe various Mal-ID evaluation results starting with the *Mal-ID basic* model followed by the results of two enhancements aimed to improve Mal-ID performance.

#### 3.3.1 Results of Mal-ID basic Model

Table 3 presents the detection performance of the proposed method for 70% of the benign files and 90% of the malware files that are used for training.



**Trend Micro, Inc.**
**EX1019-82**

| TPR | FPR | PPV | NPV | Accuracy | AUC | BCR | BER |
|---|---|---|---|---|---|---|---|
| 0.909 | 0.006 | 0.944 | 0.99 | 0.986 | 0.951 | 0.952 | 0.048 |

Table 3: Predictive Performance of *Mal-ID basic*.

Kolter and Maloof (2006) conducted rigorous research to find the best combination of n-gram length, n-gram number, features selection and classification method. They reported that the combination of five hundred 4-grams, gain ratio feature selection and boosted decision tree (AdaBoost.M1 with J48 as a base classifier) produced excellent results where the AUC was over 0.99. As you recall, we reproduced the work of Kolter and Maloof (gain ratio, 500 4-grams with boosted decision tree; referred to as *GR500BDT*) to objectively compare the performance of our methods and theirs under the same conditions such as dataset content, dataset training size, etc. A preliminary evaluation indicated that Rotation Forest (RF) boosting method (Rodriguez et al., 2006) performed better than AdaBoost.M1 and many other non-boosting methods such as J48, therefore RF was selected for our evaluation. The results of the evaluation are presented in Table 4 below.

| Method | Features | Feature selection | FPR | TPR | Acc | AUC |
|---|---|---|---|---|---|---|
| *GR500BDT* | 4grams | Gain Ratio | **0.094** | **0.959** | 0.948 | 0.929 |
| Mal-ID | Mal-ID | - | **0.006** | **0.909** | **0.986** | **0.951** |

Table 4: Comparison between Mal-ID and *GR500BDT*. The best results are colored with blue and the worst results are colored in red.

### 3.3.2 Results of combining Mal-ID with ML generated models

As you recall we attempted to improve the *Mal-ID basic* method by using Mal-ID features with various classifiers. The following figures show comparison of various detection performance measures. Many detection performance measures were recorded and reported as presented in the figures below. Please note that "TrainPercentage" refers to the percentage of benign datasets and ranges from 30 to 70 percent. Malware dataset percentages range from 40 to 90 percent. The ratio between malware and benign was kept fixed for all cases.

Figure 4 reports the average cross-entropy for a classifier by averaging the entropy of the posteriori probability that it outputs to all test instances. As expected, we see that the cross-entropy decreases as the training set size increases. For the largest training set, Mal-ID basic shows the best decrease in a posteriori cross-entropy.

Figure 5 presents the accuracy of the *Mal-ID basic* model as well that of the *Mal-IDF+NN* and *Mal-IDF+RF* models. As expected, the accuracy increases almost linearly as the training set size increases. For small training set sizes, *Mal-IDF+RF* outperforms the other methods. However, for the largest training set, the *Mal-ID basic* model eventually achieves the best results.

Figure 6 presents the TPR of all methods. *Mal-IDF+C4.5* demonstrates the lowest TPR. The *Mal-IDF+NN* and *Mal-IDF+RF* models perform the best. The *Mal-ID basic* model benefits the most from increasing the training set size. In small training sets, the difference between the *Mal-ID basic* model and either *Mal-IDF+NN* or *Mal-IDF+RF*



**Trend Micro, Inc.**
**EX1019-83**

are statistically significant. However, for larger training sets the differences are no longer significant.

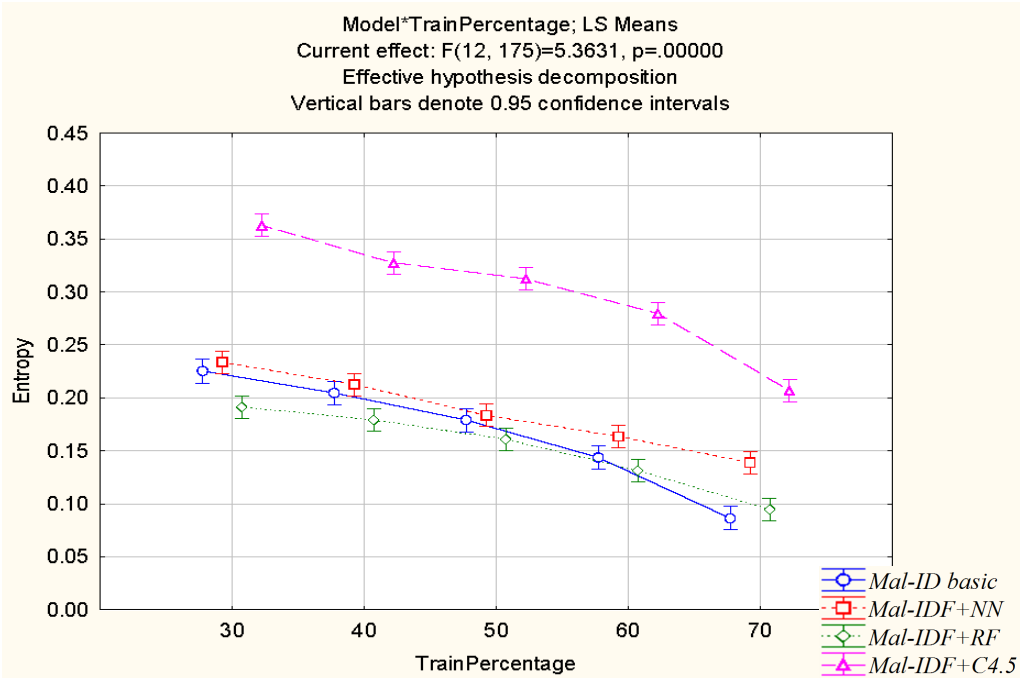


Figure 4: Comparing the a posteriori cross-entropy of various detection modules as a function of training set percentage increase.

Figure 7 presents the FPR of all methods. The *Mal-ID basic* model demonstrates the best performance. *Mal-IDF+C4.5*, on the other hand, demonstrates the lowest FPR. The performance of *Mal-IDF+NN* does not improve as the training set increases. The *Mal-ID basic* model significantly outperforms *Mal-IDF+C4.5* and *Mal-IDF+NN*. Additionally, a paired t-test indicates the *Mal-ID basic*'s FPR is significantly lower than the FPR of *Mal-IDF+RF* with $p < 0.0001$.

Figure 8 presents the area under the ROC curve for the *Mal-ID basic* model, *Mal-IDF+NN* and *Mal-IDF+RF*. All models improve as the training set increases. The *Mal-ID basic* model shows the lowest AUC but also benefits the most from increasing the training set size. The lower AUC of the *Mal-ID basic* model can be explained by the fact that contrary to the other models, the *Mal-ID basic* model is a *discrete* classifier. Discrete classifiers produce only a single point in ROC space (Fawcett, 2004) and therefore their calculated AUC appears lower.

When we examined the balanced error rate (BER) for *Mal-ID basic*, *Mal-IDF+NN* and *Mal-IDF+RF* Models, we noticed that the BER measure decreases for all models as the training set increases. *Mal-ID basic* demonstrated a significant and sharp decline in the BER as the training set increases. In almost all cases, the *Mal-IDF+RF* achieved the lowest BER. With the largest training set there is no significant difference between the *Mal-ID basic* model and the *Mal-IDF+RF* model.



**Trend Micro, Inc.**
**EX1019-84**

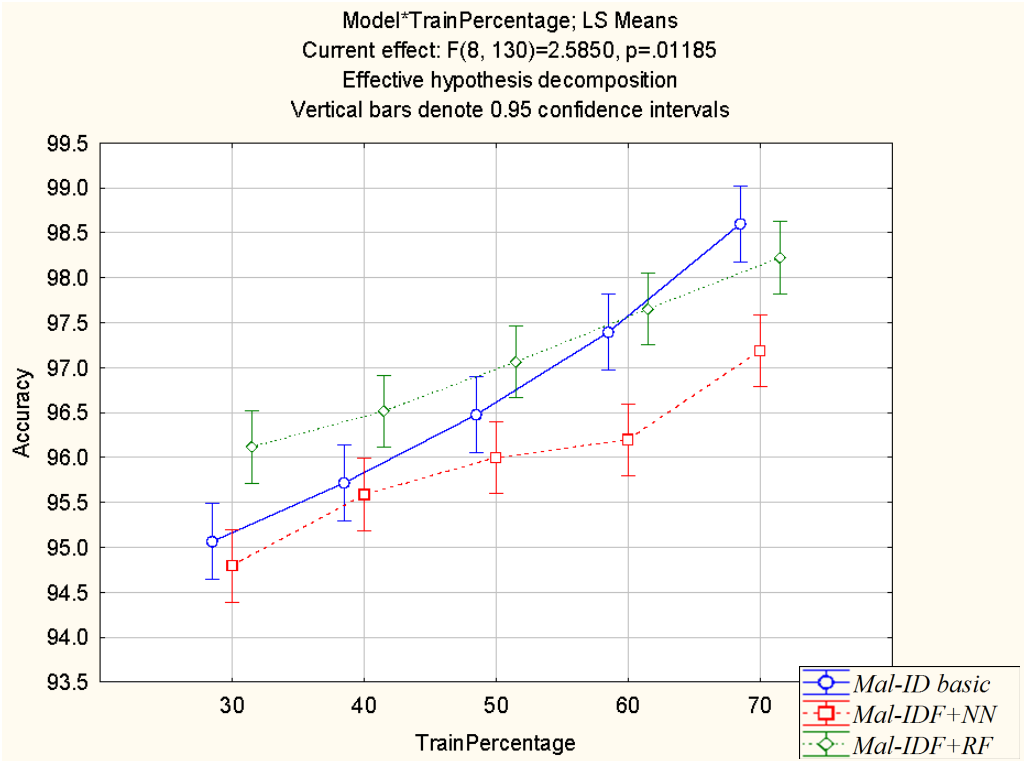


Figure 5: Comparing the accuracy performance of the *Mal-ID basic* model with the machine learning methods on various training set size percentages.

When we compared the NPV of the *Mal-ID basic* model with the NPV of the *Mal-IDF+NN* and *Mal-IDF+RF*, we noticed, as expected, that the NPV increases almost linearly as the training set size increases. For small training set sizes, *Mal-IDF+RF* and *Mal-IDF+NN* outperform the other methods. Eventually, however, there is no statistically significant difference for the largest training set.

When we compared the PPV of the *Mal-ID basic* model with the PPV of the *Mal-IDF+NN*, *Mal-IDF+C4.5* and *Mal-IDF+RF*, we found out that *Mal-ID basic* has the best PPV for all training set sizes. The *Mal-IDF+RF* performed better than the *Mal-IDF+NN* and the *Mal-IDF+NN* performed better than *Mal-IDF+C4.5*.

To sum up, in many cases Mal-ID basic outperforms the methods that use Mal-ID features combined with a ML classifier and we conclude that a simple decision rule is sufficient.

### 3.3.3 Combining Mal-ID with ML models post processing

As you recall, we have attempted to improve the *Mal-ID basic* method by using the method to zero-patch the benign common library parts. To measure and compare the effect of the Mal-ID patching prior to classifying, we preformed an evaluation using four methods: *GR500BDT*, *Mal-IDF+GR500BDT*, *Mal-ID basic*, and *Mal-IDF+RF*.



**Trend Micro, Inc.**
**EX1019-85**

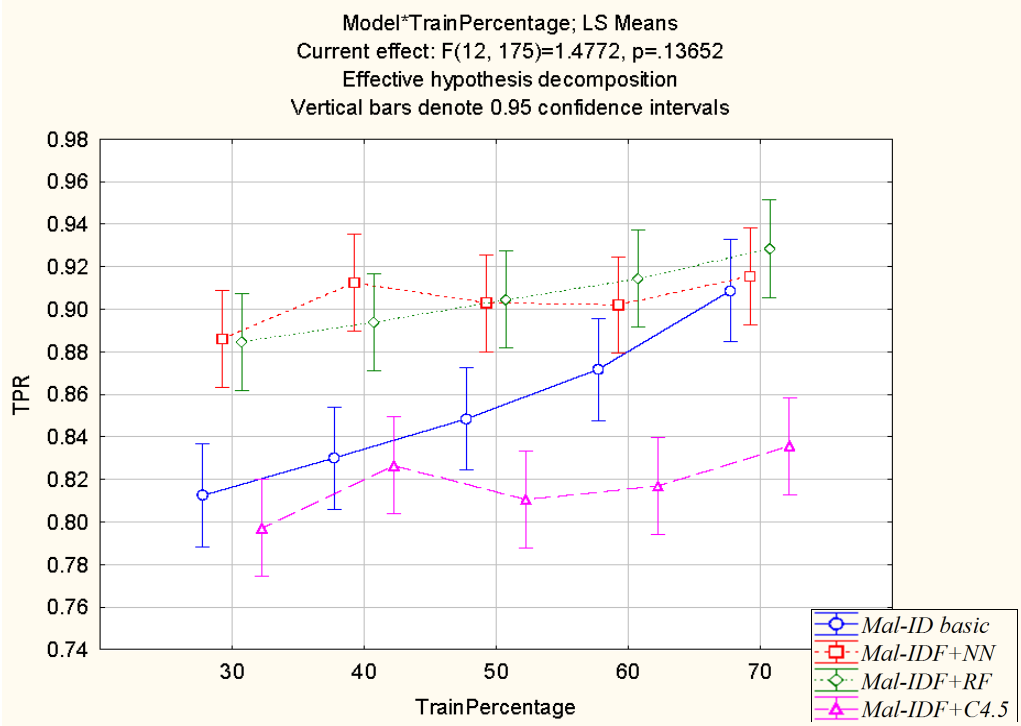


Figure 6: Comparing the true positive rate of the *Mal-ID basic* model with the machine learning methods on various training set size percentages.

Figure 9 compares the accuracy performance using various training set sizes. The results show that with *Mal-IDP+GR500BDT* we were able to improve performance but only on relatively small training sets. However, compared to the known *GR500BDT*, *Mal-IDP+GR500BDT* show significant and consistent improvements in accuracy by about 2%. All Mal-ID variations were able to outperform *GR500BDT* regardless of training set size. It should be noted that on the one hand we should have expected to an improvement in the predictive performance when the training set size increases. On the other hand because we also increase the imbalance ratio between benign and malware therefore we should have expected to a decrease in the predictive performance. Eventually we observe that accuracy of GR*500BDT* remains almost constant.

Figure 10 compares FPR performance under various training set sizes. The results indicate that there is slight but constant improvement in terms of FPR when first performing a patch with Mal-ID (*Mal-IDP+GR500BDT*) instead of using n-gram without patching (*GR500BDT*). The performance of all n-gram-based methods decreases sharply when the training set consists of more than 50% benign files. The graph shows that in terms of FPR, the *Mal-ID basic* method always performs slightly better than the *Mal-IDF+RF* method and both methods perform significantly better than n-gram based methods. In other words, the graph shows that in terms of FPR, there is a significant difference between methods that utilize n-gram features and those that utilize the Mal-ID meta-features.



**Trend Micro, Inc.**
**EX1019-86**

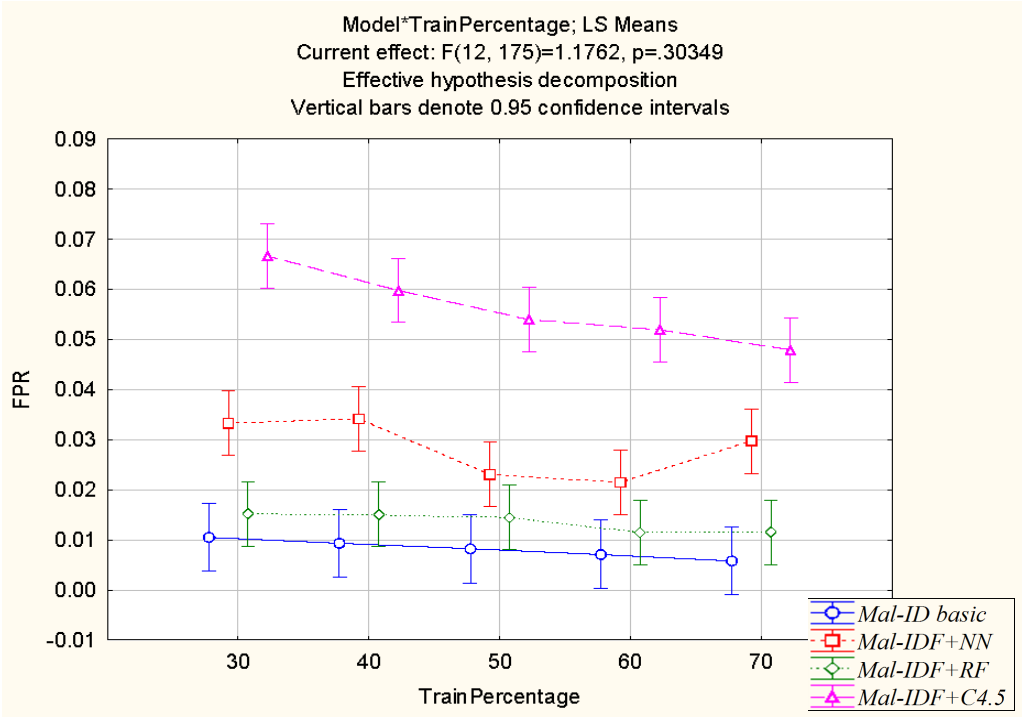


Figure 7: Comparing the false positive rate of the *Mal-ID basic* model with the machine learning methods on various training set size percentages.

Table 5 summarizes the detection performance results for the various Mal-ID methods and the *GR500BDT* baseline and can help in choosing the best method when considering detection performance only. Other important considerations will be discussed below. The results demonstrate that *Mal-IDP+GR500BDT* always outperforms *GR500BDT* baseline and *Mal-IDP+GR500BDT* should be used when the highest TPR is desired and a high FPR is acceptable. However *Mal-ID basic* and *Mal-IDF+RF* seems to be the best choice for more balanced performance with extremely low FPR and for achieving the highest accuracy.

| **Method** | **Feature selection** | **FPR** | **TPR** | **Acc** | **AUC** |
|---|---|---|---|---|---|
| *GR500BDT* (un-patched + RF) | Gain Ratio | 0.094 | 0.959 | 0.948 | 0.929 |
| *Mal-IDP+GR500BDT* (patched + RF) | Gain Ratio | 0.093 | **0.977** | 0.963 | 0.946 |
| *Mal-ID basic* | Mal-ID | **0.006** | 0.909 | **0.986** | 0.951 |
| *Mal-IDF+RF* (Mal-ID features + RF) | None | **0.006** | 0.916 | 0.985 | **0.995** |

Table 5: A comparison of various Mal-ID methods and RF when using maximum training size. The best results are colored with blue.



**Trend Micro, Inc.**
**EX1019-87**

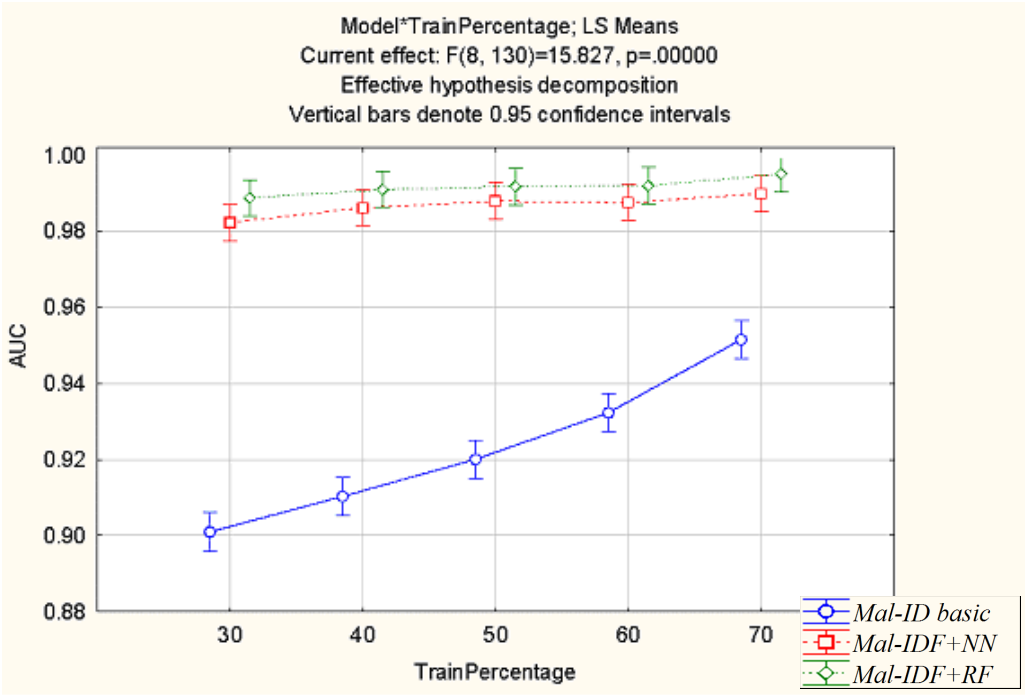


Figure 8: Comparing the AUC of the *Mal-ID basic* model with the machine learning methods on various training set size percentages.

Table 6 presents the training time (in seconds) and detection time (in ms) of all examined methods. The evaluation computer used an Intel Q6850 CPU with 4GB of RAM. All times were measured using only 1 CPU core. The training time of Mal-ID based methods does not include building the CFL and TFL which took around 30 seconds. As expected the training time increases with the training size. In addition, GR500BDT training time does not include the n-gram feature extraction and selection (which took more than ten minutes). The Mal-*ID basic* and Mal-IDF+C4.5 methods demonstrated the best training time performance with less than one second. The detection time seems almost constant regardless of training set size. The only exception is Mal-IDF+RF in which detection time increases almost linearly as the training set increases. Note that the size of the trees (number of nodes) which constitute the rotation forest usually increases with the training set. This can be explained by the fact that the number of leaves in the tree is bounded by the training set size. Larger trees require a longer traversal time and features calculation. Recall that in rotation forest, the features used in the various nodes are linear combination of the original features.

Table 7 reports the mean TPR of *Mal-ID basic* for small malwares (size<=350K) and large malware (size>350K) using the largest training set. Note that the FPR is kept as reported in Table 5 (i.e. FPR=0.006). The results show that the TPR for both small



**Trend Micro, Inc.**
**EX1019-88**

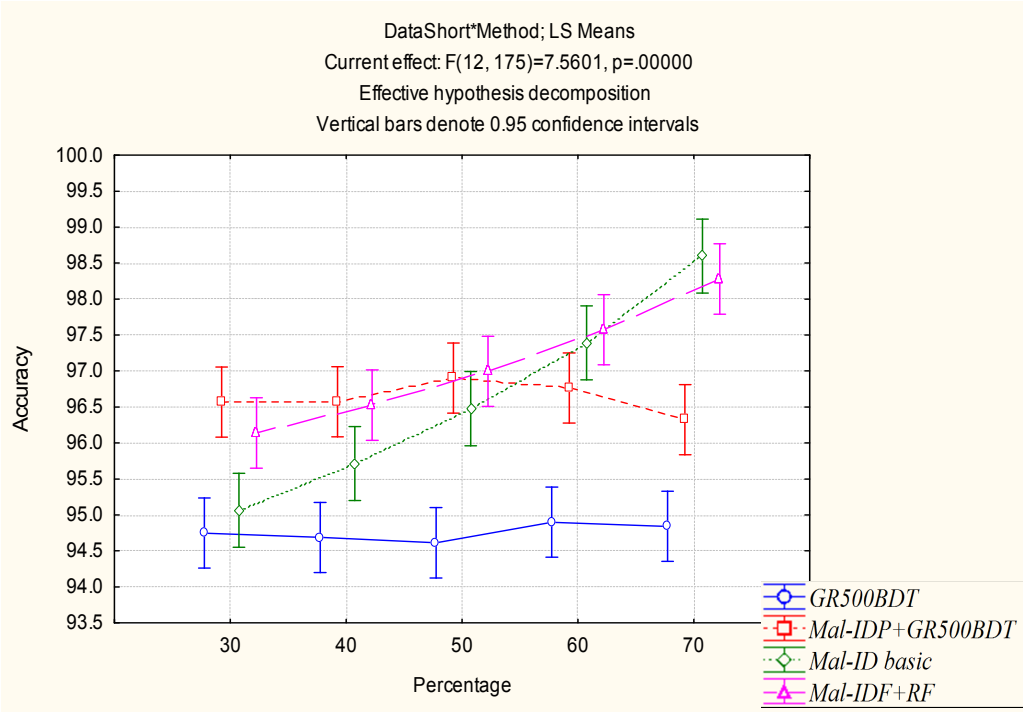


Figure 9: Comparing the accuracy of various Mal-ID-based methods and the n-gram method on various training set size percentages.

| | | **Train Percentage** | | | | |
|---|---|---|---|---|---|---|
| | **Method** | **30** | **40** | **50** | **60** | **70** |
| Training Time (in sec) | MalID-Basic | 0.05 | 0.08 | 0.11 | 0.15 | 0.21 |
| | Mal-IDF+RF | 17.19 | 26.00 | 36.35 | 45.78 | 83.50 |
| | Mal-IDF+C4.5 | 0.12 | 0.17 | 0.22 | 0.33 | 0.43 |
| | Mal-IDF+NN | 24.33 | 32.16 | 40.33 | 48.37 | 56.93 |
| | GR500BDT | 21.74 | 34.91 | 59.86 | 64.88 | 75.19 |
| | Mal-IDP+GR500BDT | 20.93 | 31.42 | 42.96 | 55.65 | 63.43 |
| Detection Time per file (in ms) | MalID-Basic | 27.86 | 27.86 | 27.86 | 27.86 | 27.86 |
| | Mal-IDF+RF | 49.17 | 54.69 | 63.66 | 73.95 | 95.82 |
| | Mal-IDF+C4.5 | 27.86 | 27.86 | 27.86 | 27.86 | 27.86 |
| | Mal-IDF+NN | 27.92 | 27.92 | 27.90 | 27.89 | 27.88 |
| | GR500BDT | 29.63 | 29.83 | 29.83 | 29.85 | 29.83 |
| | Mal-IDP+GR500BDT | 29.01 | 29.01 | 29.02 | 28.98 | 28.97 |

Table 6: Training and Detection Time.

and large group is very similar indicating that MAL ID is not affected by the size of the examined malware.



**Trend Micro, Inc.**
**EX1019-89**

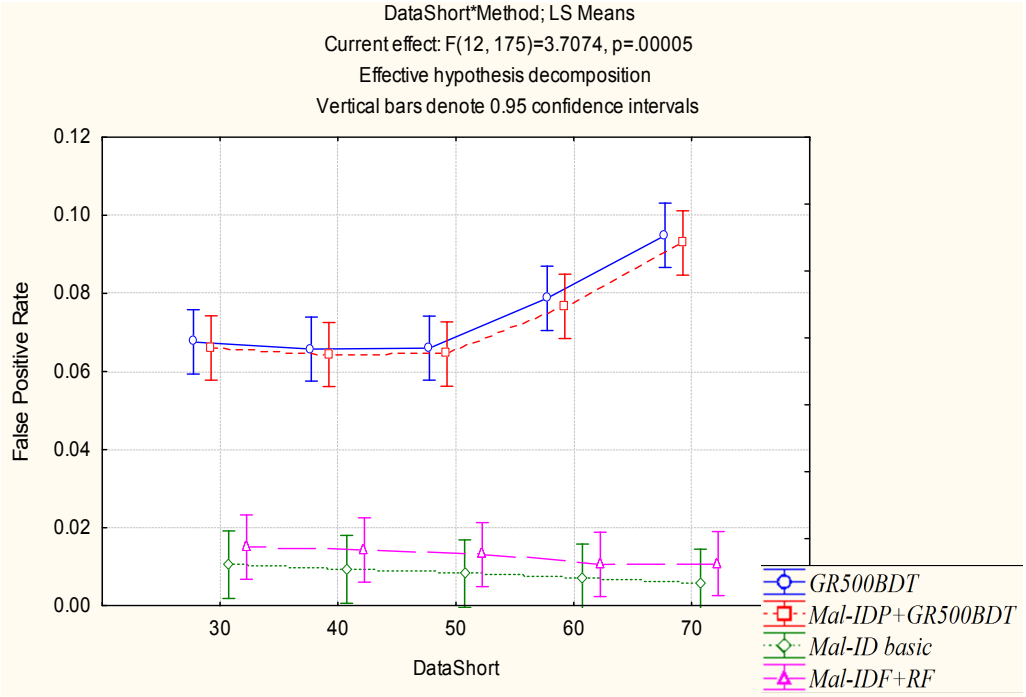


Figure 10: Comparing the FPR of various Mal-ID-based methods and the n-gram method on various training set size percentages.

| Malware Size | TPR | Number of Malwares | Mean Size |
|---|---|---|---|
| Small | 0.909 | 675 | 96K |
| Large | 0.908 | 174 | 554K |

Table 7: A comparison of TPR (True Positive Rate) Mal-ID basic for small and large malwares when using maximum training size.

In order to estimate the effect of obfuscation on detection rate, we have divided the tested malware into two groups –obfuscated and non-obfuscated. Because we were not informed which executable was obfuscated, we have used the following method. We compressed the executables using Zip and sorted them according to the compression ratio. We used a threshold of 50% compression ratio to decide which executable is probably obfuscated. The selection of this threshold was based on experiments of compressing non- obfuscated executables. According to this threshold, about 37.5% of the malware are considered to be obfuscated. Table 8 reports the mean TPR of *Mal-ID basic* for obfuscated and non-obfuscated groups using the largest training set. Note that the FPR is kept as reported in Table 5 (i.e. FPR=0.006). The results show that the TPR for both obfuscated and non-obfuscated group is very similar with slight advantage to detecting obfuscated malwares.



**Trend Micro, Inc.**
**EX1019-90**

There might be several reasons that can explain why the TPR of obfuscated binaries appears to be higher than the TPR of non-obfuscated binaries. One reason can be that many obfuscated malwares are generated by automated tools that have distinctive properties. For example, malware developers are sharing tools for facilitating the generation of new malwares. For example, in the web site http://vx.netlux.org/, one can find many tools (such as Falckon Encrypter that is used for obfuscation) that can be used by the malware developers but are not used by benign software developers. All malware that utilize the Falckon Encrypter, share the same decryption segment.

| **Malware type** | **TPR** | **Mean Compression Ratio** |
|---|---|---|
| Obfuscated | 0.932 | 41% |
| Non-obfuscated | 0.893 | 62% |

Table 8: A comparison of TPR (True Positive Rate) Mal-ID basic for obfuscated and non-obfuscated malware when using maximum training size.

## 4. Discussion

This paper proposes a new approach for automatically detecting executable malwares of all kinds and sizes. The results show that using the *Mal-ID basic* and other Mal-ID variants are useful in detecting malware. As can be seen from Table 3, the *Mal-ID basic* method performs very well in respect to all measures. Not only is the accuracy very high (0.986), but the FPR is remarkably low (0.006). In addition, the low Mal-ID BER indicates that the errors are almost uniformly distributed among the malicious and benign files.

As explained in Section 3.3.1, we choose to implement *GR500BDT* as a baseline for comparing the performance of the *Mal-ID basic* method. *GR500BDT* is very similar to the method proposed by Kolter and Maloof (2006). The evaluation shows that *GR500BDT* performed well, but was unable to achieve the AUC of 0.995 that Kolter and Maloof reported. This was probably due to differences in dataset content, training size, the benign and malware ratio and possibly other factors. As can be seen from Table 4, under identical conditions the Mal-ID methodology was able to outperform *GR500BDT* in terms of FPR, accuracy and AUC. The FPR of *GR500BDT* method came to almost 10%; Mal-ID FPR was more than 15 times lower.

Once it was established that the *Mal-ID basic* method performs well (in fact better than the best baseline method) we wanted to examine Mal-ID behavior with different train sizes to test if *Mal-ID basic* performs in a stable and "trustworthy" manner. In addition, it was interesting to determine if combining *Mal-ID basic* with ML-generated models, as explained in Section 3.3.2, would yield a better performing malware detection method.

The results presented in Figure 4 to Figure 8 show that combining Mal-ID with ML-based models enabled us to improve many aspects of the *Mal-ID basic* method when training sets are not maximal. However, as training set size increases, the benefit of combining *Mal-ID basic* with ML-based models diminishes. At maximal training set size, the *Mal-ID basic* method almost always demonstrates the best performance or a performance that is



**Trend Micro, Inc.**
**EX1019-91**

statistically equal to the combined methods. It is also important to note that, contrary to the other methods, all measures that we implemented indicated that the *Mal-ID basic* method benefited the most from training set increase and always performed in an expected manner. Thus, it may be considered more stable and "trustworthy" than the other methods.

It is interesting to note that while the performance of non-n-gram methods (*Mal-ID basic* and *Mal-IDF+RF*) continues to improve as more training data become available, the n-gram based methods show a sharp decrease in performance in terms of FPR (see Figure 10). This can be explained by the fact that n-gram methods induce relatively simple patterns that can be learned with comparatively small training sets (30%). The potential benefit of additional training data is nullified by the undesirable increase in the probability that relevant n-gram will be mistakenly considered as non-contributing features. In fact, it is well known that decision trees increase their chances of overfitting when they have more nodes. But in order to have more nodes, they need a larger training set. Thus a larger dataset might increase the chance of overfitting especially in cases were there are many irrelevant and noisy features.

The comparison of our two additional methods, *Mal-IDF+RF* and *Mal-IDP+GR500BDT*, with a *GR500BDT* baseline is very important in proving the validity of Mal-ID itself and explaining its excellent performance:

1. (a) Under identical conditions, boosted decision tree, operating on *Mal-ID basic* meta-features (*Mal-IDF+RF*), outperformed boosted decision tree operating on n-gram (*GR500BDT*). The comparison suggests that Mal-ID meta-features are useful in contributing to malware detection and probably more meaningful than simple n-gram in capturing a file's essence.

   (b) Under identical conditions, boosted decision tree operating on *Mal-ID basic* patched files (*Mal-IDP+GR500BDT*) outperformed boosted decision tree operating on non-patched files (*GR500BDT*). The comparison suggests that the novel Mal-ID common segment analysis approach is better than the common approach that treats files as black boxes or which interprets files PE header only.

Since *Mal-ID basic* and *Mal-IDF+RF* methods benefit from both more meaningful features and common segment analysis, they are able to achieve a better overall performance than state-of-the-art *GR500BDT*.

Considering detection performance only when choosing a malware detection method may not be enough; it is important to consider other aspects as well.

### 4.1 Model interpretability

Mal-ID basic uses only one static interpretable classification model and therefore experts in the field can be more confident when accepting or rejecting a classification. For instance, once *Mal-ID basic* has detected a yet unknown malware, it is possible to support or reject the classification. The reason is that each detected segment, that passed the Mal-ID filter stage as explained in Section 2, can be tracked back to a specific malware or malware group. Moreover, the specific offset location were the segments appear can be examined to determine the precise nature of the threat, if any exists. Disassembly or reverse engineering of the whole malware is no longer required. Even without examining the segment code, one



**Trend Micro, Inc.**
**EX1019-92**

can make an educated guess about the nature of the threat by examining the list of known malwares that the segment appears in. The other methods do not provide such benefits.

### 4.2 Incremental

As more malwares are discovered, it is important to update the models from time to time. With *Mal-ID basic* it is particularly easy. Since the model is static, no reconstruction is necessary; all that is required is to just to add or subtract files from the TFL. The CFL can be updated in a similar manner.

### 4.3 Anytime Detection

Recall that both *Mal-ID basic* and *Mal-IDF+RF* operates on segments. Because *Mal-ID basic* and *Mal-IDF+RF* use relatively large segments and the model is not comprised of combined features from the whole file, it is possible to stop detection at anytime during file scan and determine if the scanned part is malicious. n-gram-based methods are not designed to diagnose part of file but rather whole files only.

### 4.4 Default signature for real-time malware detection hardware

The end result of applying *Mal-ID basic* method is a file segment or segments that appear in malware files only and thus may be used as a signature for anti-virus tools. The detected malware segments can be used, as described by Filiol (2006), to generate signatures resistant against black-box analysis. Moreover, because *Mal-ID basic* produces a simple signature and has *anytime detection traits*, the signature can be used with commercially available real-time intrusion prevention systems (IPS). IPSs require the *anytime detection trait* to act as real-time malware filtering devices and thus promote and provide users with default protection. Having both malware detection and signature generation could help shorten the window of vulnerability. Tahan et al. (2010) have presented a methodology with complete implementation for automatic signature generation, utilizing similar and compatible techniques, which archived excellent results in the evaluation. Thus, the method presented by Tahan et al. (2010) can be easily adopted to produce signature upon detection for the solution presented in this paper.

### 4.5 Large files scalability

Nowadays it's quite common to embed large resources such as JPEG pictures and small animations into executables. This inflation is also true for malware. It is estimated[1] that the mean malware size has increased from 150K (in 2005) to 350K (in 2010). As files become larger, the effectiveness of classification with small n-gram should decrease due to the increase in file entropy. In other words, the more n-gram with equal appearance probability, the greater the misclassification probability becomes. Since *Mal-ID basic* and *Mal-IDF+RF* use relatively large segments (64 bytes) and in addition filter-out high entropy parts, they should be less susceptible to misclassification caused by large files or files with high entropy traits. Figure 10 shows that the Mal-ID methods that operate on large segments (of 64

1. $http://nakedsecurity.sophos.com/2010/07/27/large-piece-malware/$



**Trend Micro, Inc.**
**EX1019-93**

bytes) has less FPR misclassification then the method that operated on small n-gram (of 4 bytes). We further examined this hypothesis in Table 7.

### 4.6 Analysis of Mal-ID performance on Obfuscated Malware

Based on the results presented so far, we hypothesize that the proposed Mal-ID method performs well in a mixed environment where both obfuscated (including compressed or encrypted) and plain executable files exist. In this sense, we referred to malware as they are found "in the Wild".

The results of Table 8 agree with the previously-made observation that ML techniques can classify malware that are obfuscated (compressed or encrypted or both). For example, Kolter and Maloof (2006) have noted that ML can detect obfuscated malware. In this paper, we have independently reconfirmed the validity of the above observation using our method. In this experiment, we succeeded to keep FPR relatively low (FPR=0.006), however it should be noted that this value was obtained when our corpus contained 2,627 benign files and 849 malware files (i.e. a benign to malware ratio of 3:1). In reality this ratio can be much higher and therefore one should expect to obtain elevated FPR values.

There seem to be previously suggested explanations to this phenomenon. According to Kolter and Maloof (2006), the success in detecting obfuscated malware relies on learning certain forms of obfuscation such as run-time decompression. Kolter and Maloof (2006) conclude that "...this does not seem problematic as long as those forms are correlated with malicious executables".

Additional explanations can be suggested to the ability to identify obfuscated malware. Studies such as that presented by Newsome and Song (2005), or by Newsome et al. (2005) noticed that in many cases malware requires fixed sequences to be used in the body of the malware (which must exist before self-decryption or self-decompression) in order to exploit a specific vulnerability and self-propagate. Such fixed sequences can be used for detection. This might explain the success in detecting obfuscated malware.

Because the performance of MAL ID is achieved with no disassembly, Op-Code analysis, executable header analysis, unpacking nor any other preprocessing, we hypothesize that the method should be scalable to other Operating Systems and hardware types. Still one can think on cases where preprocessing will be required. Theoretically an attacker can specifically design a malware that will make it hard for MAL ID to detect it. In particular, if a malware is designed such that the entropy measure will be high for all segments, it will be undiscovered by the Mal-ID basic method. In this case Mal-ID can be extended by incorporating an unpacker operating before it, such as those that are incorporated into anti-viruses tools (Kasparsky). However, similar to Kolter and Maloof (2006), we decided to evaluate the raw power of our methods without any use of an unpacker.

## 5. Summary and Future Work

In this paper we have described novel methods based on machine learning to detect malware in executable files without any need for preprocessing the executables. The basic method that we presented works on the segment level for detecting new malware instead of using the entire file as usually done in machine learning based techniques. The *Mal-ID basic* method and its derived variants were rigorously tested to evaluate their effectiveness under



**Trend Micro, Inc.**
**EX1019-94**

many conditions using a wide variety of measures. The results demonstrate the effectiveness of the methods. In all cases, most of the performance measures showed that the proposed methods significantly outperformed the baseline method *GR500BDT* which is known for its excellent performance (Kolter and Maloof, 2004, 2006). For each method we have pinpointed its strong points and suggested cases where it should be preferred over the others.

We believe this study has made several contributions to malware detection research, including the introduction of:

1. a new and effective method for malware detection based on common segment analysis and supporting algorithms. The importance of common segment analysis to the process of malware detection was identified and demonstrated. The results suggest the method can boost performance for many methods that utilize n-gram.

2. new kinds of features - *Mal-ID basic* meta-features. The results suggest that the meta-features are much more effective than the commonly used n-gram and probably more meaningful in terms of file representation. We believe that *Mal-ID basic* meta-features could inspire many kinds of additional meta-features that could prove useful.

3. BCR, BER, PPV, NPV and entropy decrease for measuring the performance of malware detection methods. Using these measures, in addition to the commonly utilized performance measures (TPR, FPR, accuracy and AUC), is not generally practiced. However, these features are helpful in describing the behavior of a new method, particularly when it is not possible to compare results under identical settings and dataset imbalance.

The results also indicate that by extracting meaningful features, it is sufficient to employ one simple detection rule for classifying unknown executables.

In the future, we aim to examine the effect of systematically collecting and choosing the benign file set on the performance of the proposed methods. In the evaluations that were conducted for this study, the benign file set was collected randomly and the files used may have had a large degree of similarity. It is our assumption that systematically collecting and choosing common segments will provide a better representation of benign common segments and a more robust and lower FPR. A robust and low FPR will enable the use of more sensitive malware detection methods (or parameters that affect malware detection) without increasing the FPR too much. As a result, we hope to see further increase in the AUC measure. Finally the Mal-ID basic method was developed as a crisp classifier. Additional research is required for developing a method for ranking the examined files according to their presumed threat level. One straightforward measure is the ratio between the segments found in the TFL and the segments found in the CFL. In addition, it will be interesting to test the proposed method on live network data and on an institutional network and determine if it detects malware that is not detected by other means. Finally, future work may repeat the evaluation Mal-ID on a larger scale with thousands of malware samples and tens of thousands of non-malware samples. For this purpose, we might need to upscale software components to accommodate large dataset and suitable hardware. In addition, in order to use the proposed method in practice by the industry, fine tuning of the various parameters might be required.



**Trend Micro, Inc.**
**EX1019-95**

Additional studies might be needed to fully evaluate the performance of Mal-ID under various obfuscation scheme, including use of recursive unpacking. In this paper we focused only on "pure" Mal-ID methods and therefore we did not investigate the proper means to incorporate unpacker.

## Acknowledgments

The authors gratefully thank the action editor, Prof. Charles Elkan and the anonymous reviewers whose constructive comments considerably strengthened this manuscript.

## References

T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 41–42. IEEE, 2004.

D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

R.E. Bellman, R.E. Bellman, R.E. Bellman, and R.E. Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, 1966.

C.M. Bishop. Neural networks for pattern recognition. 1995.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

R. Caruana, T. Joachims, and L. Backstrom. Kdd-cup 2004: results and analysis. *ACM SIGKDD Explorations Newsletter*, 6(2):95–108, 2004.

J. Dai, R. Guha, and J. Lee. Efficient virus detection using dynamic instruction sequences. *Journal of Computers*, 4(5):405–414, 2009.

G. Demiröz and H. Güvenir. Classification by voting feature intervals. *Machine Learning: ECML-97*, pages 85–92, 1997.

Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer. Applying machine learning techniques for detection of malicious code in network traffic. *KI 2007: Advances in Artificial Intelligence*, pages 44–50, 2007.

T. Fawcett. Roc graphs: Notes and practical considerations for researchers. *ReCALL*, 31 (HPL-2003-4):1–38, 2004.

Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029, 1993.

E. Filiol. Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, 2(1):35–50, 2006.



**Trend Micro, Inc.**
**EX1019-96**

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for large-scale risk minimization. *The Journal of Machine Learning Research*, 10:2157–2192, 2009.

Y. Freund and R.E. Schapire. A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1401–1406. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.

T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531, 1999.

O. Henchiri and N. Japkowicz. A feature selection and evaluation scheme for computer virus detection. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 891–895. IEEE, 2006.

R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

Thorsten Joachims. Making large-scale svm learning practical. *Advances in Kernel Methods Support Vector Learning*, pages 169–184, 1999.

G.H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the eleventh conference on uncertainty in artificial intelligence*, volume 1, pages 338–345. San Mateo, 1995.

Jeremy Z. Kolter and Marcus A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 470–478, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1.

J.Z. Kolter and M.A. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744, 2006.

E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, 53(4):1483–1494, 2009.

T. Mitchell. *Machine learning*. McGraw-Hill, 1997.

J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226–241. IEEE, 2005.

N. O'Farrell. Cybercrime costs society over 1 trillion, 2011.

J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32(2):245–257, 1987.



**Trend Micro, Inc.**
**EX1019-97**

J.R. Quinlan. *C4. 5: programs for machine learning.* Morgan kaufmann, 1993.

J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006.

L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

L. Rokach, R. Romano, and O. Maimon. Negation recognition in medical narrative reports. *Information Retrieval*, 11(6):499–538, 2008.

M.G. Schultz, E. Eskin, F. Zadok, and S.J. Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.

Symantec. Symantec global internet security threat report trends for 2010, 2010.

G. Tahan, C. Glezer, Y. Elovici, and L. Rokach. Auto-sign: an automatic signature generator for high-speed malware filtering devices. *Journal in computer virology*, 6(2):91–103, 2010.

C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145. IEEE, 1999.

I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang. An intelligent pe-malware detection system based on association mining. *Journal in computer virology*, 4(4):323–334, 2008.

Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5(4):283–293, 2009.

Y. Ye, T. Li, K. Huang, Q. Jiang, and Y. Chen. Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list. *Journal of Intelligent Information Systems*, 35(1):1–20, 2010.

S. Yu, S. Zhou, L. Liu, R. Yang, and J. Luo. Detecting malware variants by byte frequency. *Journal of Networks*, 6(4):638–645, 2011.

B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang. Malicious codes detection based on ensemble learning. *Autonomic and trusted computing*, pages 468–477, 2007.



**Trend Micro, Inc.**
**EX1019-98**