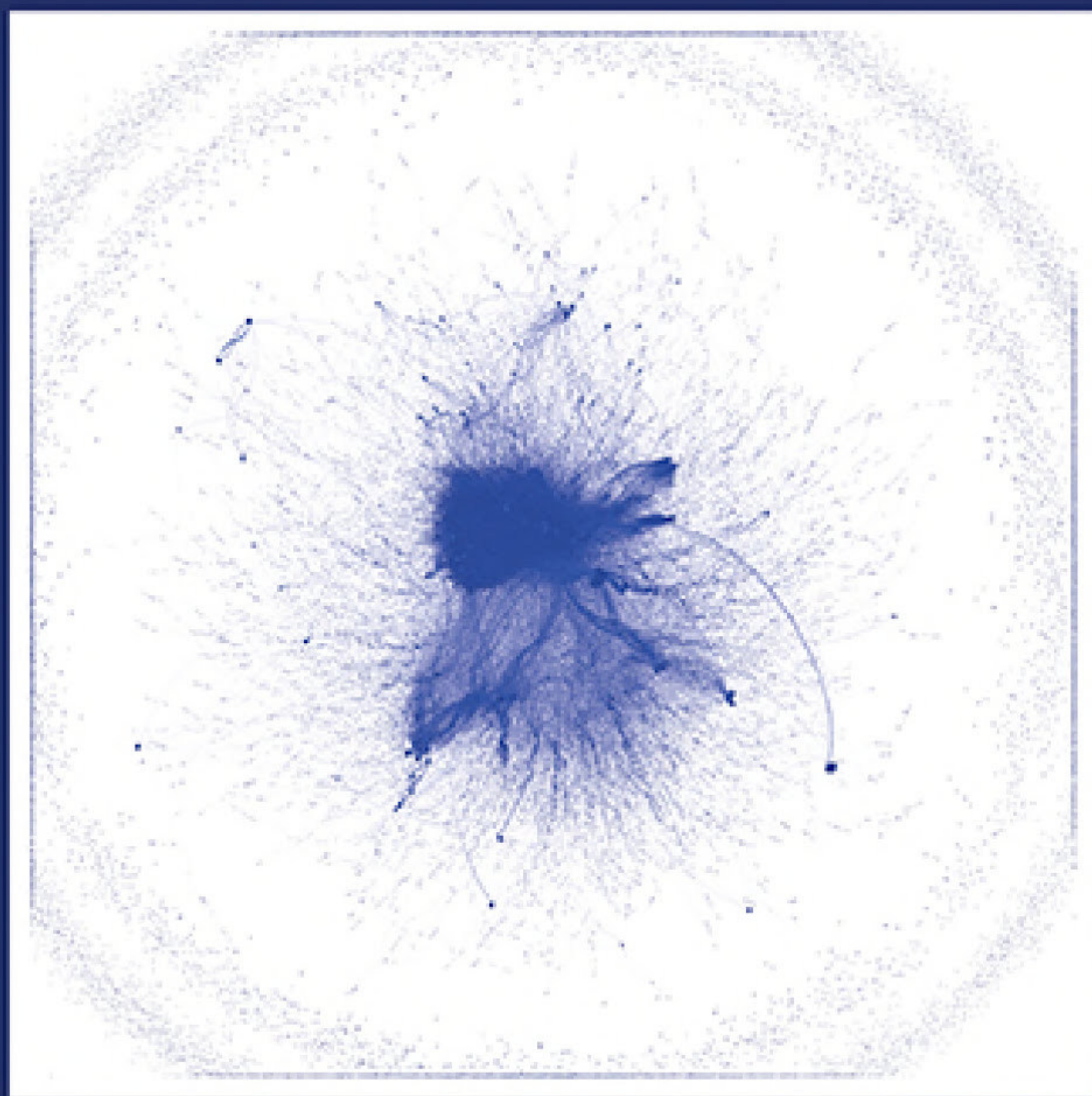



Security Science and Technology – Vol. 3

# Data Science for Cyber-Security



Nick Heard • Niall Adams  
Patrick Rubin-Delanchy • Melissa Turcotte  
*editors*

 World Scientific

# Data Science for Cyber-Security

*Editors*

**Nick Heard**

Imperial College London, UK

**Niall Adams**

Imperial College London, UK

**Patrick Rubin-Delanchy**

University of Bristol, UK

**Melissa Turcotte**

Los Alamos National Laboratory, USA

*Published by*

World Scientific Publishing Europe Ltd.

57 Shelton Street, Covent Garden, London WC2H 9HE

*Head office:* 5 Toh Tuck Link, Singapore 596224

*USA office:* 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

### **Library of Congress Cataloging-in-Publication Data**

Names: Heard, Nicholas, editor. | Rubin-Delanchy, Patrick, editor. | Turcotte, Melissa, editor.

Title: Data science for cyber-security / edited by Nick Heard (Imperial College London, UK),  
Niall Adams (Imperial College London, UK), Patrick Rubin-Delanchy (University of Bristol, UK),  
Melissa Turcotte (Los Alamos National Laboratory, USA).

Description: [Hackensack] New Jersey : World Scientific, [2018] | Series: Security science and  
technology ; volume 3 | Includes bibliographical references and index.

Identifiers: LCCN 2018021228 | ISBN 9781786345639 (hc : alk. paper)

Subjects: LCSH: Internet--Security measures--Data processing--Congresses. |  
Data protection--Statistical methods--Congresses.

Classification: LCC TK5105.59 .D383 2018 | DDC 005.8--dc23

LC record available at <https://lcn.loc.gov/2018021228>

### **British Library Cataloguing-in-Publication Data**

A catalogue record for this book is available from the British Library.

Copyright © 2019 by World Scientific Publishing Europe Ltd.

*All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the Publisher.*

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

For any available supplementary material, please visit

<https://www.worldscientific.com/worldscibooks/10.1142/Q0167#t=suppl>

Desk Editors: Anthony Alexander/Jennifer Brough/Shi Ying Koe

Typeset by Stallion Press

Email: [enquiries@stallionpress.com](mailto:enquiries@stallionpress.com)

Printed in Singapore

# Contents

<i>Preface</i>	v
1. Unified Host and Network Data Set <i>Melissa J. M. Turcotte, Alexander D. Kent and Curtis Hash</i>	1
2. Computational Statistics and Mathematics for Cyber-Security <i>David J. Marchette</i>	23
3. Bayesian Activity Modelling for Network Flow Data <i>Henry Clausen, Mark Briers and Niall M. Adams</i>	55
4. Towards Generalisable Network Threat Detection <i>Blake Anderson, Martin Vejman, David McGrew and Subharthi Paul</i>	77
5. Feature Trade-Off Analysis for Reconnaissance Detection <i>Harsha Kumara Kalutarage and Siraj Ahmed Shaikh</i>	95
6. Anomaly Detection on User-Agent Strings <i>Eirini Spyropoulou, Jordan Noble and Christoforos Anagnostopoulos</i>	127

7.	Discovery of the Twitter Bursty Botnet	145
	<i>Juan Echeverria, Christoph Besel and Shi Zhou</i>	
8.	Stochastic Block Models as an Unsupervised Approach to Detect Botnet-Infected Clusters in Networked Data	161
	<i>Mark Patrick Roeling and Geoff Nicholls</i>	
9.	Classification of Red Team Authentication Events in an Enterprise Network	179
	<i>John M. Conroy</i>	
10.	Weakly Supervised Learning: How to Engineer Labels for Machine Learning in Cyber-Security	195
	<i>Christoforos Anagnostopoulos</i>	
11.	Large-scale Analogue Measurements and Analysis for Cyber-Security	227
	<i>George Cybenko and Gil M. Raz</i>	
12.	Fraud Detection by Stacking Cost-Sensitive Decision Trees	251
	<i>Alejandro Correa Bahnsen, Sergio Villegas, Djamila Aouada and Björn Ottersten</i>	
13.	Data-Driven Decision Making for Cyber-Security	267
	<i>Mike Fisk</i>	
	<i>Index</i>	293

## Chapter 1

# Unified Host and Network Data Set

Melissa J. M. Turcotte<sup>\*,‡</sup>, Alexander D. Kent<sup>\*</sup> and Curtis Hash<sup>†</sup>

*\*Los Alamos National Laboratory,  
Los Alamos, NM 87545, USA*

*†Ernst & Young, New Mexico, USA*

*‡mturcotte@lanl.gov*

The lack of data sets derived from operational enterprise networks continues to be a critical deficiency in the cyber-security research community. Unfortunately, releasing viable data sets to the larger community is challenging for a number of reasons, primarily the difficulty of balancing security and privacy concerns against the fidelity and utility of the data. This chapter discusses the importance of cyber-security research data sets and introduces a large data set derived from the operational network environment at Los Alamos National Laboratory (LANL). The hope is that this data set and associated discussion will act as a catalyst for both new research in cyber-security as well as motivation for other organisations to release similar data sets to the community.

## 1. Introduction

The lack of diverse and useful data sets for cyber-security research continues to play a profound and limiting role within the relevant research communities and their resulting published research. Organisations are reticent to release data for security and privacy reasons. In addition, the data sets that are released are encumbered in a variety of ways, from being stripped of so much information that they no longer provide rich research and analytical opportunities, to being so constrained by access restrictions that key details are lacking and independent validation is difficult. In many cases, organisations do not collect relevant data in sufficient volumes or with high enough

fidelity to provide cyber-research value. Unfortunately, there is generally little motivation for organisations to overcome these obstacles.

In an attempt to help stimulate a larger research effort focused on operational cyber-data as well as to motivate other organisations to release useful data sets, Los Alamos National Laboratory (LANL) has released two data sets for public use (Kent, 2014, 2016). A third, entitled the *Unified Host and Network Data Set*, is introduced in this chapter.

The Unified Host and Network Data Set is a subset of network flow and computer events collected from the LANL enterprise network over the course of approximately 90 days.<sup>a</sup> The host (computer) event logs originated from the majority of LANL's computers that run the Microsoft Windows operating system. The network flow data originated from many of the internal core routers within the LANL enterprise network and are derived from router netflow records. The two data sets include many of the same computers but are not fully inclusive; the network data set includes many non-Windows computers and other network devices.

Identifying values within the data sets have been de-identified (anonymised) to protect the security of LANL's operational IT environment and the privacy of individual users. The de-identified values match across both the host and network data allowing the two data elements to be used together for analysis and research. In some cases, the values were not de-identified, including well-known network ports, system-level usernames (not associated to people) and core enterprise hosts. In addition, a small set of hosts, users and processes were combined where they represented well-known, redundant entities. This consolidation was done for both normalisation and security purposes.

In order to transform the data into a format that is useful for researchers who are not domain experts, a significant effort was made to normalise the data while minimising the artefacts that such normalisation might introduce.

### **1.1. Related public data sets**

A number of public, cyber-security relevant data sets currently are referenced in the literature (Glasser and Lindauer, 2013; Ma *et al.*, 2009) or

---

<sup>a</sup>The network flow data are only 89 days due to missing data on the first day.

are available online.<sup>b</sup> Some of these represent data collected from operational environments, while others capture specific, pseudo real-world events (for example, cyber-security training exercises). Many data sets are synthetic and created using models intended to represent specific phenomenon of relevance; for example, the Carnegie Mellon Software Engineering Institute provides several insider threat data sets that are entirely synthetic (Glasser and Lindauer, 2013). In addition, many of the data sets commonly seen within the research community are egregiously dated. The DARPA cyber-security data sets (Cyber-Systems and Technology Group, 1998) published in the 1990s are still regularly used, even though the systems, networks and attacks they represent have almost no relevance to modern computing environments.

Another issue is that many of the available data sets have restrictive access and constraints on how they may be used. For example, the U.S. Department of Homeland Security provides the Information Marketplace for Policy and Analysis of Cyber-risk and Trust (IMPACT,<sup>c</sup> which is intended to facilitate information sharing. However, the use of any of the data hosted by IMPACT requires registration and vetting prior to access. In addition, data owners may (and often do) place limitations on how and where the data may be used.

Finally, many of the existing data sets are not adequately characterised for potential researchers. It is important that researchers have a thorough understanding of the context, normalisation processes, idiosyncrasies and other aspects of the data. Ideally, researchers should have sufficiently detailed information to avoid making false assumptions and to reproduce similar data. The need for such detailed discussion around published data sets is a primary purpose of this chapter.

The remainder of this chapter is organised as follows: a description of the Network Flow Data is given in Section 2 followed by the Windows Host Log Data in Section 3. Finally, a discussion of potential research directions is given in Section 4.

---

<sup>b</sup><https://www.ll.mit.edu/ideval/data/>, <http://malware-traffic-analysis.net/>, <http://www.unb.ca/cic/research/datasets/index.html>.

<sup>c</sup><https://www.dhs.gov/csd-impact>.



## 2. Network Flow Data

The network flow data set included in this release is comprised of records describing communication events between devices connected to the LANL enterprise network. Each *flow* is an aggregate summary of a (possibly) bi-directional network communication between two network devices. The data are derived from Cisco NetFlow Version 9 (Claise, 2004) flow records exported by the core routers. As such, the records lack the payload-level data upon which most commercial intrusion detection systems are based. However, research has shown that flow-based techniques have a number of advantages and are successful at detecting a variety of malicious network behaviours (Sperotto *et al.*, 2010). Furthermore, these techniques tend to be more robust against the vagaries of attackers, because they are not searching for specific signatures (for example, byte patterns) and they are encryption-agnostic. Finally, in comparison to full-packet data, collection, analysis and archival storage of flow data at enterprise scales is straightforward and requires minimal infrastructure.

### 2.1. Collection and transformation

As mentioned previously, the raw data consisted of NetFlow V9 records that were exported from the core network routers to a centralised collection server. While V9 records can contain many different fields, only the following are considered: *StartTime*, *EndTime*, *SrcIP*, *DstIP*, *Protocol*, *SrcPort*, *DstPort*, *Packets* and *Bytes*. The specifics of the hardware and flow export protocol are largely irrelevant, as these fields are common to all network flow formats of which the authors are aware.

This data can be quite challenging to model without a thorough understanding of its various idiosyncrasies. The following paragraphs discuss two of the most relevant issues with respect to modelling. For a comprehensive overview of these issues, among others, readers can refer to Hofstede *et al.* (2014).

Firstly, note that these flow records are uni-directional (*uniflows*): each record describes a stream of packets sent from one network device (*SrcIP*) to another (*DstIP*). Hence, an established TCP connection — bi-directional by definition — between two network devices, *A* and *B*, results in two flow records: one from *A* to *B* and another from *B* to *A*. It follows that there is no relationship between the direction of a flow and the initiator of a

bi-directional connection (i.e., it is not known whether *A* or *B* connected first). This is the case for most netflow implementations as bi-directional flow (*biflow*) protocols such as Trammell and Boschi (2008) have yet to gain widespread adoption. Clearly, this presents a challenge for detection of attack behaviours, such as lateral movement, where directionality is of primary concern.

Secondly, significant duplication can occur due to flows encountering multiple netflow sensors in transit to their destination. Routers can be configured to track flows on ingress and egress, and, in more complex network topologies, a single flow can traverse multiple routers. More recently, the introduction of netflow-enabled switches and dedicated netflow appliances has exacerbated the issue. Ultimately, a single flow can result in many distinct flow records. To add further complexity, the flow records are not necessarily *exact* duplicates and their arrival times can vary considerably; these inconsistencies occur for many reasons, the particulars of which are too complex to discuss in this context.

In order to simplify the data for modelling, a transformation process known as *biflowing* or *stitching* was employed. This is a process intended to aggregate duplicates and marry the opposing unflows of bi-directional connections into a single, *directed* biflow record (Table 1). Many approaches to this problem can be found in the literature (Barbosa, 2014; Berthier *et al.*, 2010; Minarik *et al.*, 2009; Nguyen *et al.*, 2017), all of them imperfect. A straightforward approach was used that relies on simple port heuristics to

Table1: Bi-directional flow data.

Field Name	Description
<i>Time</i>	The start time of the event in epoch time format.
<i>Duration</i>	The duration of the event in seconds.
<i>SrcDevice</i>	The device that likely initiated the event.
<i>DstDevice</i>	The receiving device.
<i>Protocol</i>	The protocol number.
<i>SrcPort</i>	The port used by the <i>SrcDevice</i> .
<i>DstPort</i>	The port used by the <i>DstDevice</i> .
<i>SrcPackets</i>	The number of packets the <i>SrcDevice</i> sent during the event.
<i>DstPackets</i>	The number of packets the <i>DstDevice</i> sent during the event.
<i>SrcBytes</i>	The number of bytes the <i>SrcDevice</i> sent during the event.
<i>DstBytes</i>	The number of bytes the <i>DstDevice</i> sent during the event.

decide direction. These heuristics are based on the assumption that *SrcPorts* are generally *ephemeral* (i.e., they are selected from a predefined, high range by the operating system), while *DstPorts* tend to have lower numbers that correspond to established, shared network services and will therefore be observed more frequently than ephemeral ports. The heuristics are given below in order of precedence.

- Destination ports are less than 1024 and source ports are not.
- The top 90 most frequently observed ports are destination ports.
- The smaller of the two ports is the destination port.

Each uniflow was transformed into a biflow by renaming the *Packets* and *Bytes* fields to *SrcPackets* and *SrcBytes*, respectively. *DstPackets* and *DstBytes* fields were added with initial values of zero. Next, the port heuristics were considered and, if any were violated or ambiguous, the *Src* and *Dst* attributes were swapped, effectively reversing the direction. Finally, the *5-tuple* was extracted from each record and used as the key in a lookup table.

#### *SrcIP, DstIP, SrcPort, DstPort, Protocol*

If a match was found, the flows were aggregated by keeping the minimum *StartTime*, maximum *EndTime* and summing the other attributes. If no match was found, the flow was simply added to the table. This process was performed in a streaming fashion on all of the records in the order in which they were received by the collector. Flows were periodically evicted from the lookup table after 30 minutes of inactivity (i.e., failing to match with any incoming flows). Flows that remained active for long periods of time were reported approximately every 3 hours, but were *not* evicted from the table until inactive.

While biflowing the data mitigates the problems posed by duplicates and ambiguous directionality, it does not address another significant obstacle: the lack of stable identifiers upon which to build models. In some cases, IP addresses are transient (e.g., Dynamic Host Configuration Protocol (DHCP), Virtual Private Network (VPN)). In other cases, devices have multiple IP addresses (e.g., multihoming) or one IP address is shared by multiple devices (e.g., load-balancing, NAT). Whatever the case may be, modelling the behaviour of IP addresses on a typical network is clearly

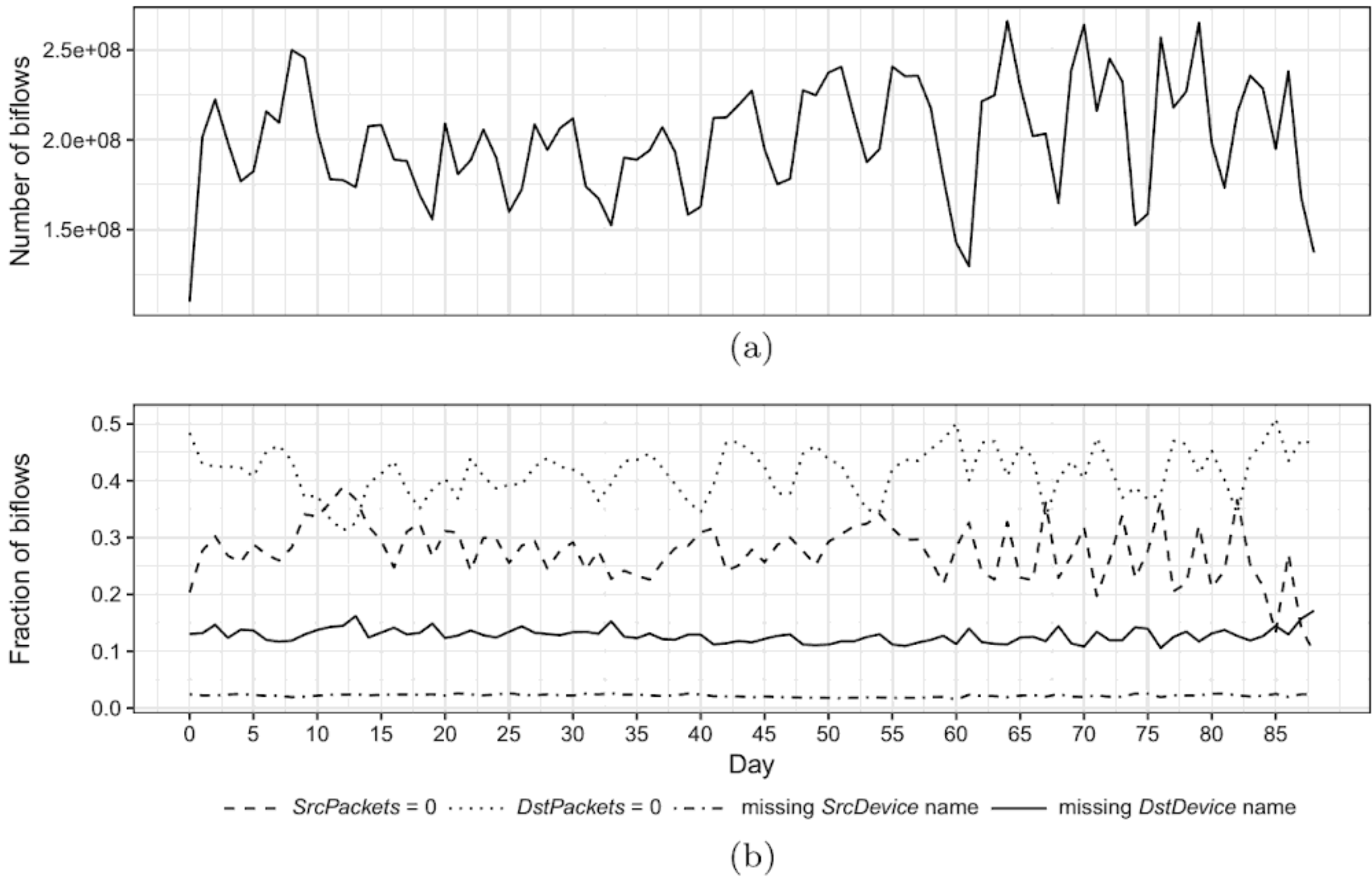


Fig. 1: (a) Daily count of biflows by end time. (b) Fraction of biflows where  $SrcPackets = 0$ ,  $DstPackets = 0$ ,  $SrcDevice$  FQDN-mapping failed and  $DstDevice$  FQDN-mapping failed.

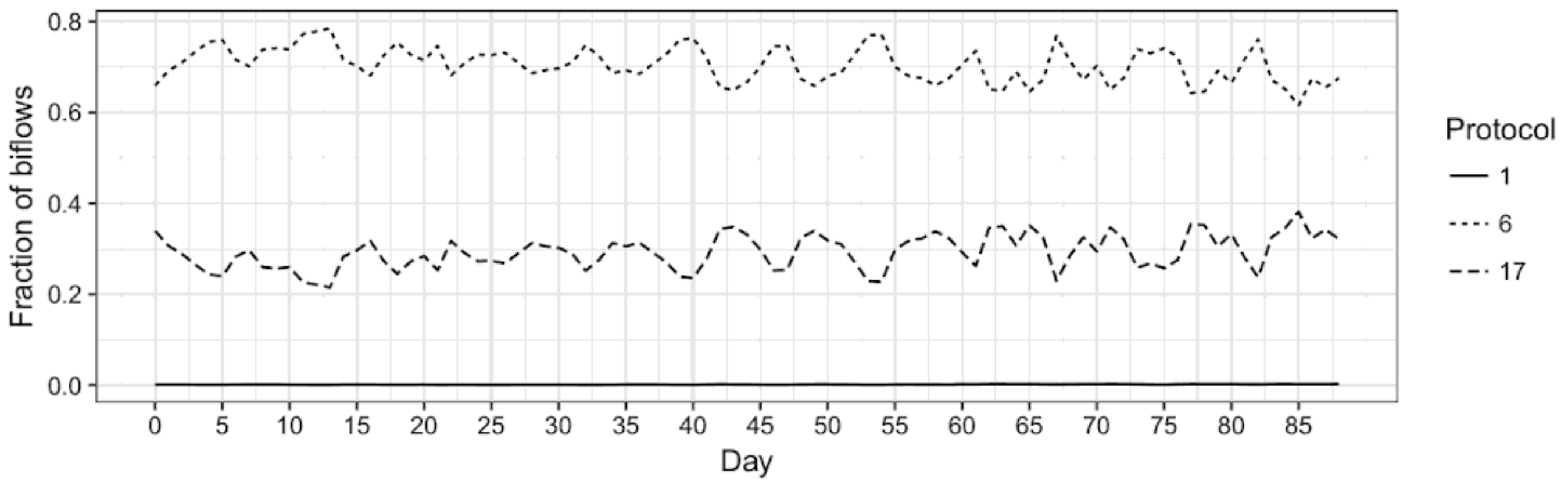


Fig. 2: Daily proportions of each *Protocol*.

error prone. Instead, one should endeavour to map IP addresses to more stable identifiers such as media access control (MAC) addresses or fully-qualified domain names (FQDN), interchangeably referred to as hostnames throughout the rest of the chapter. As with directionality, there is no perfect solution to this problem. The most appropriate identifier will depend greatly on the configuration of the target network, as well as the availability of auxiliary data sources from which a mapping can be constructed. An ideal solution will likely involve some combination of supplementary network

data (e.g., Domain Name Service (DNS) logs, DNS zone transfers, DHCP logs, VPN logs, NAC logs), business rules and considerable trial and error.

For this data release, a combination of DNS and DHCP logs was used to construct a mapping of IP addresses to FQDNs over time. The IP addresses in each biflow were then replaced with their corresponding FQDNs at the time of the flow. Where a given IP address and timestamp mapped to multiple FQDNs, business rules were incorporated to give preference to the least-ephemeral name. IP addresses that failed to map to any FQDN were left as is. The resulting mix of names and IP addresses correspond to the *SrcDevice* and *DstDevice* fields in the final data.

Finally, the data were de-identified by mapping *SrcDevice*, *DstDevice*, *SrcPort* and *DstPort* to random identifiers. In the event that the IP-to-FQDN mapping failed, the random identifier was prepended with “IP”. Well-known ports were not de-identified. Records with protocol numbers other than 6 (TCP), 17 (UDP) and (1) ICMP were removed entirely. The output from this process is provided in CSV format, one record per line, with fields in the order shown in Table 1.

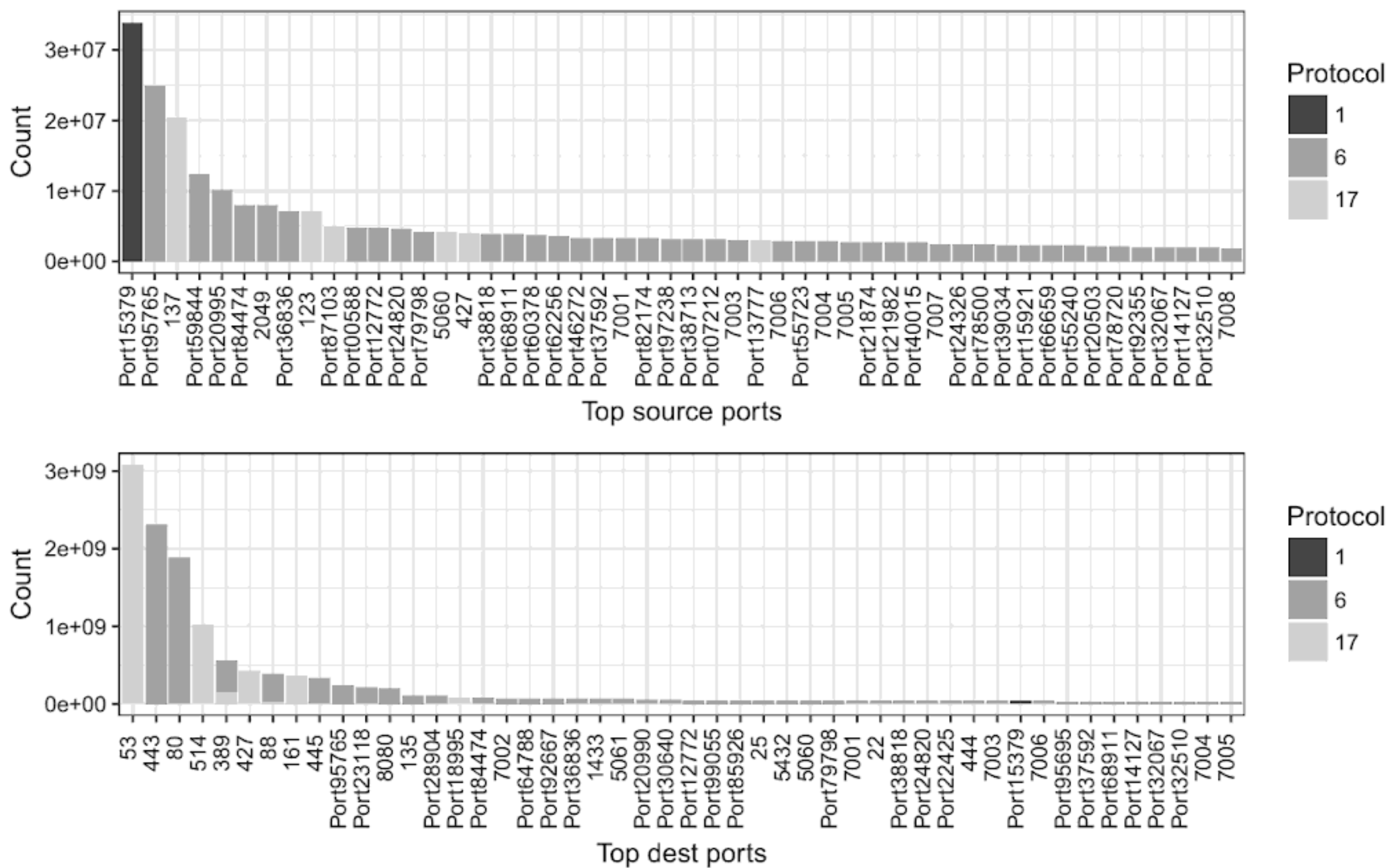


Fig. 3: Histogram of the top 50 *SrcPorts* and *DstPorts*.

## 2.2. Data quality

Several figures have been provided in order to assess the quality of the network flow data set. The top plot in Figure 1, which shows the number of biflows over time, demonstrates the periodicity that one would expect for data whose volume is driven by the comings and goings of employees during a typical 5-day workweek.

The bottom plot of Figure 1 is intended to measure the success rate of the biflowing and IP-to-FQDN mapping processes. TCP biflows where either *SrcPackets* or *DstPackets* is zero suggests a failure to find matching unflows for both directions of the exchange. Fifty Seven percent of TCP and approximately 70% of all biflows fall within this category. This can largely be attributed to LANL's netflow sensor infrastructure, which has been specifically configured to export only one direction on many routes. In addition, some devices — namely vulnerability scanners and the like — attempt to connect to all possible IP addresses within a range; this results in a significant number of unflows for which no response is possible. Likely for the same reason, IP-to-FQDN mapping failed for significantly more *DstDevices* than *SrcDevices*.

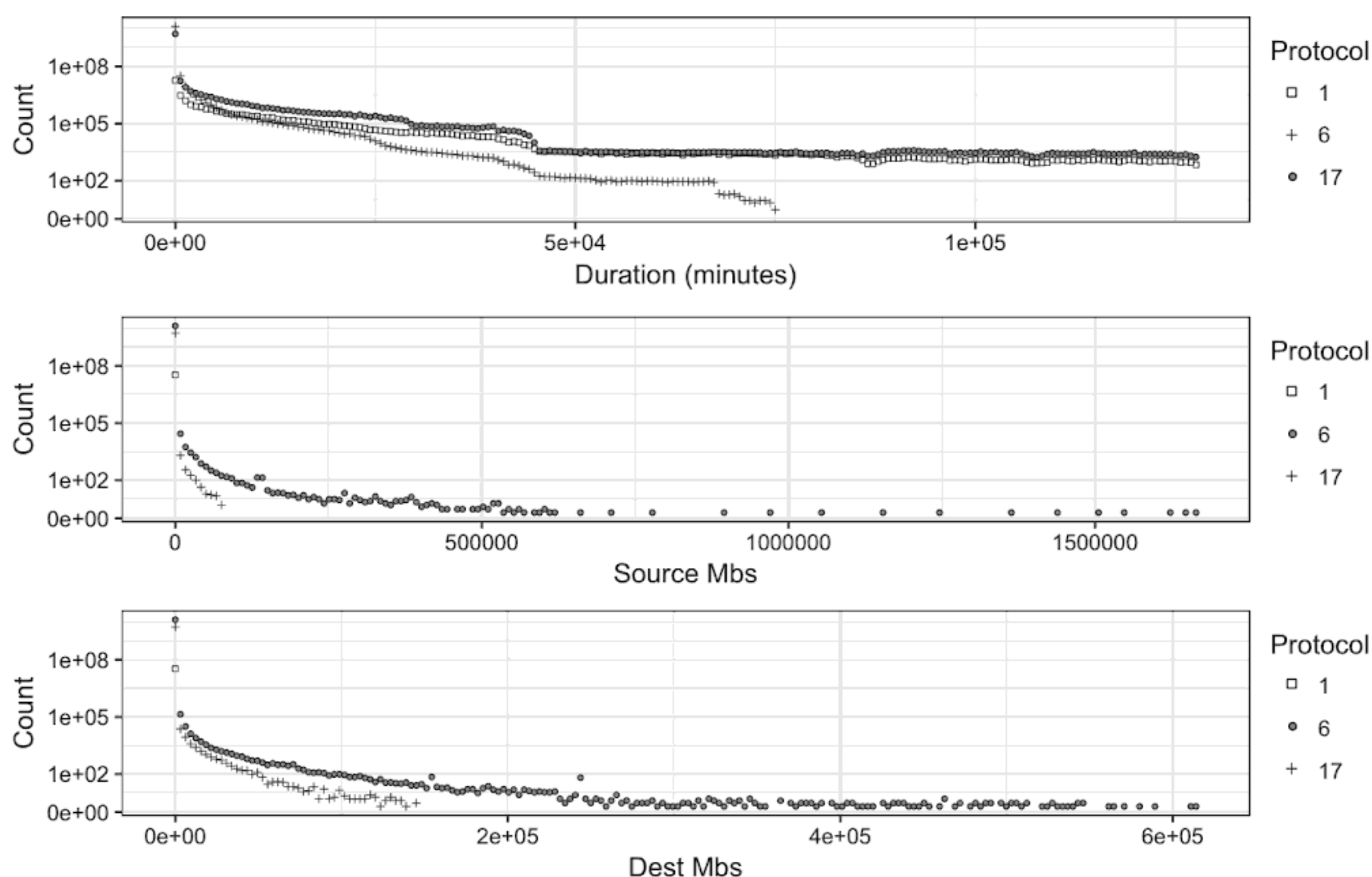


Fig. 4: Distribution of *Duration*, *SrcBytes* and *DstBytes*.

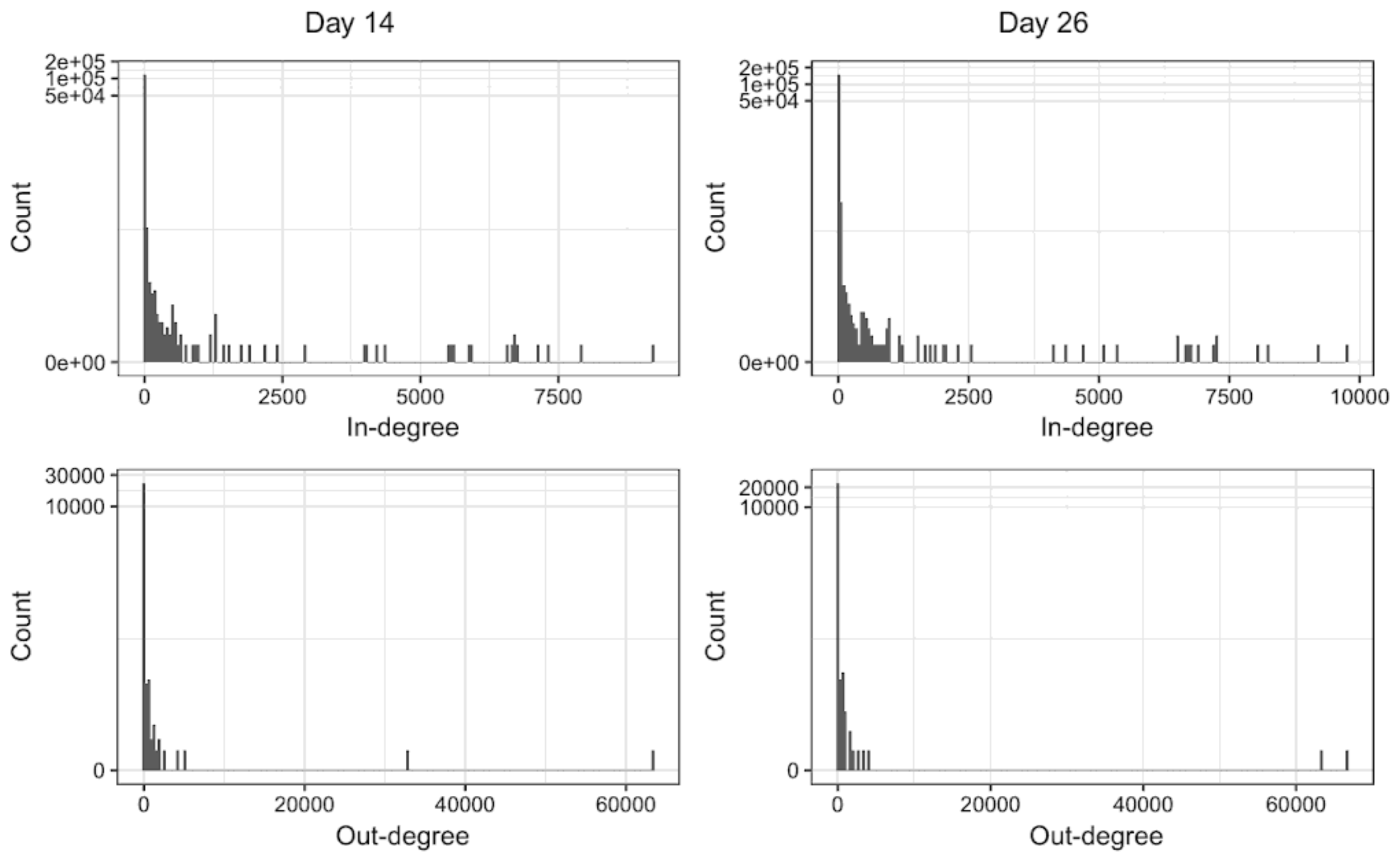


Fig. 5: In-degree and out-degree distribution for two randomly-selected days.

Figure 2 shows the daily proportion of biflows corresponding to each *Protocol*. Figure 3 contains two histograms of the top *SrcPorts* and *DstPorts* respectively. Note the non-uniformity in the *SrcPort* histogram; this illustrates either a consistent failure of the biflowing process to choose the appropriate direction or the presence of protocols that use non-ephemeral source ports. For example, the network time protocol (NTP) uses port 123 for both the source and destination ports per the specification.

Figure 4 shows the distribution of *Duration*, *SrcBytes* and *DstBytes* per *Protocol*. Of particular interest is the presence of many long-lived UDP and ICMP biflows in the data. This indicates frequent, persistent UDP and ICMP traffic sharing the same *5-tuple* and is an unfortunate side effect of not limiting the biflow transformation to TCP unflows. Finally, Figure 5 shows exemplar in-degree and out-degree distributions for two randomly-selected days.

### 3. Windows Host Log Data

As remote attackers and malicious insiders increasingly use encryption, network-only detection mechanisms are becoming less effective, particularly those that require the inspection of payload data within the network

traffic. As a result, cyber-defenders now rely heavily on endpoint agents and host event logs to detect and investigate incidents. Host event logs capture nuanced details for a wide range of activities; however, given the vast number of logged events and their specificity to an individual host, human analysts struggle to discover the few useful log entries amid the huge number of innocuous entries. Statistical analytics for host event data are in their infancy. Advanced analytical capabilities on this host data, including computer and user profiling, which move beyond signature-based methods, will increase network awareness and detection of advanced cyber-threats.

The host event data set is a subset of host event logs collected from all computers running the Microsoft Windows operating system on LANL's enterprise network. The host logs were collected with windows logging service (WLS), which is a Windows service that forwards event logs, along with administrator-defined contextual data to a set of collection servers.<sup>d</sup> The released data are in JSON format in order to preserve the structure of the original events, unlike the two previously released data sets based on this log source (Kent, 2014, 2016). The events from the host logs included in the data set are all related to authentication and process activity on each machine.

Table 2 contains the subset of *EventIDs* included from the event logs in the released data set and a brief description of each; a more detailed description is available online.<sup>e</sup> Figure 6 shows the percentage of *EventIDs* contained in the logs, as well as the *LogonTypes* for *EventIDs* 4624, 4625 and 4634.

Each record in the data set will have some of the event attributes listed in Appendix A and Table B.1 specifies which *EventIDs* have each attribute. Note that not all events with a given *EventID* share the same set of attributes. If an expected attribute was missing from the original host log record, then the attribute was not included in the corresponding record in the de-identified data set.

All records will contain the attributes *EventID*, *LogHost* and *Time*. *LogHost* indicates the network host where the record was logged. For

---

<sup>d</sup><http://honeywell.com/sites/aero-kcp/SiteCollectionDocuments/WindowsLoggingServiceSummary.pdf>.

<sup>e</sup><https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx>.



Table 2: Host log *EventIDs*.

<i>EventID</i>	Description	
Authentication events		
4768	Kerberos authentication ticket was requested (TGT)	
4769	Kerberos service ticket was requested (TGS)	
4770	Kerberos service ticket was renewed	
4774	An account was mapped for logon	
4776	Domain controller attempted to validate credentials	
4624	An account successfully logged on, see Logon Types	
4625	An account failed to logon, see Logon Types	
4634	An account was logged off, see Logon Types	
4647	User initiated logoff	
4648	A logon was attempted using explicit credentials	
4672	Special privileges assigned to a new logon	
4800	The workstation was locked	
4801	The workstation was unlocked	
4802	The screensaver was invoked	
4803	The screensaver was dismissed	
Process events		
4688	Process start	
4689	Process end	
System events		
4608	Windows is starting up	
4609	Windows is shutting down	
1100	Event logging service has shut down (often recorded instead of <i>EventID</i> 4609)	
<i>LogonTypes</i> ( <i>EventIDs</i> : 4624, 4625 and 4634)		
2 — Interactive	5 — Service	9 — New Credentials
3 — Network	7 — Unlock	10 — Remote Interactive
4 — Batch	8 — Network Clear Text	11 — Cached Interactive
12 — Cached Remote-Interactive	0 — Used only by the system account	

directed authentication events, this attribute will always correspond to the computer to which the user is authenticating, and the source computer will be given by *Source*. For the user associated with the record, if the *UserName* ends in \$ then it will correspond to the *computer account* for the specified computer. These computer accounts are host-specific accounts within the

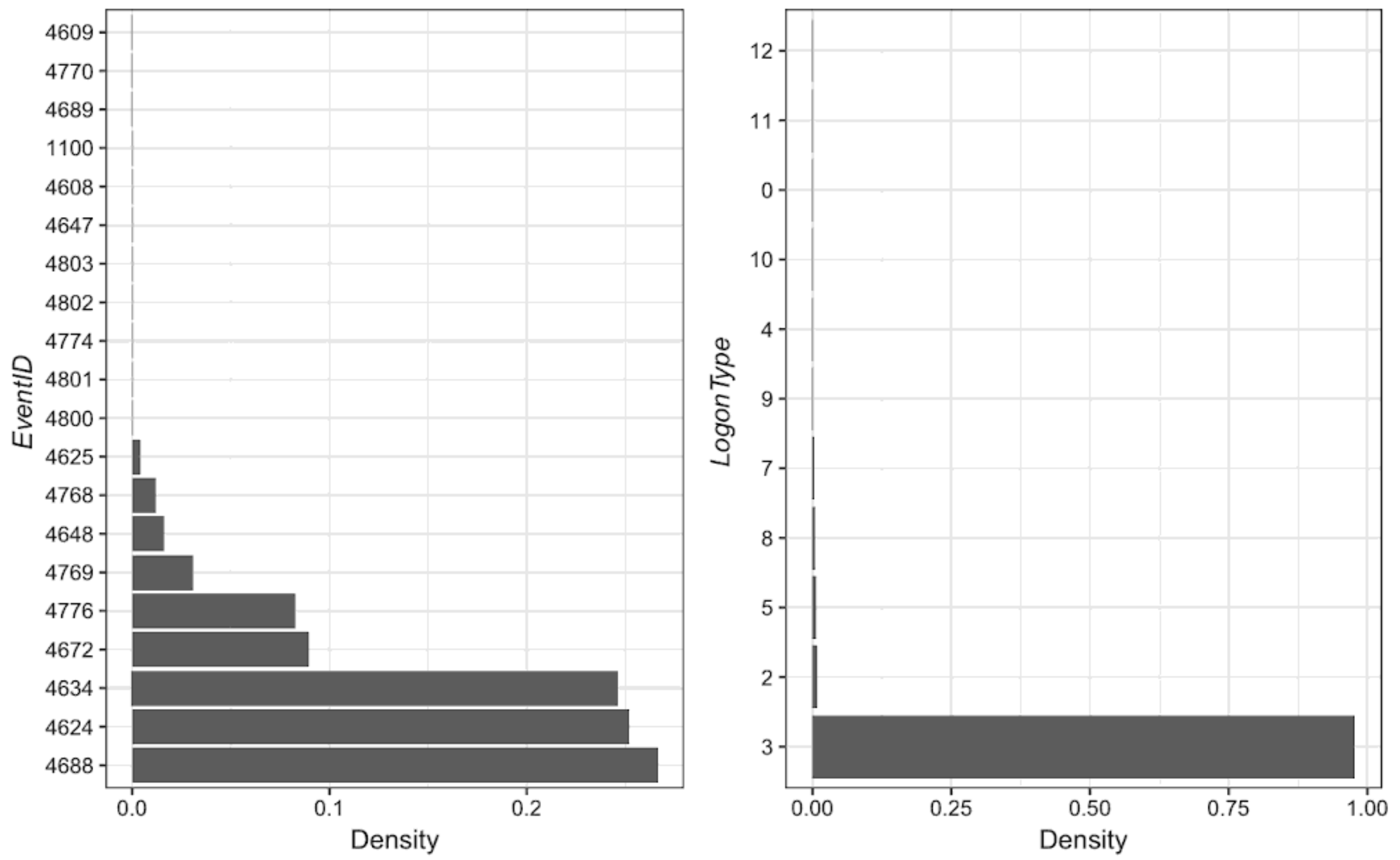


Fig. 6: Histogram of the *EventIDs* and *LogonTypes*.

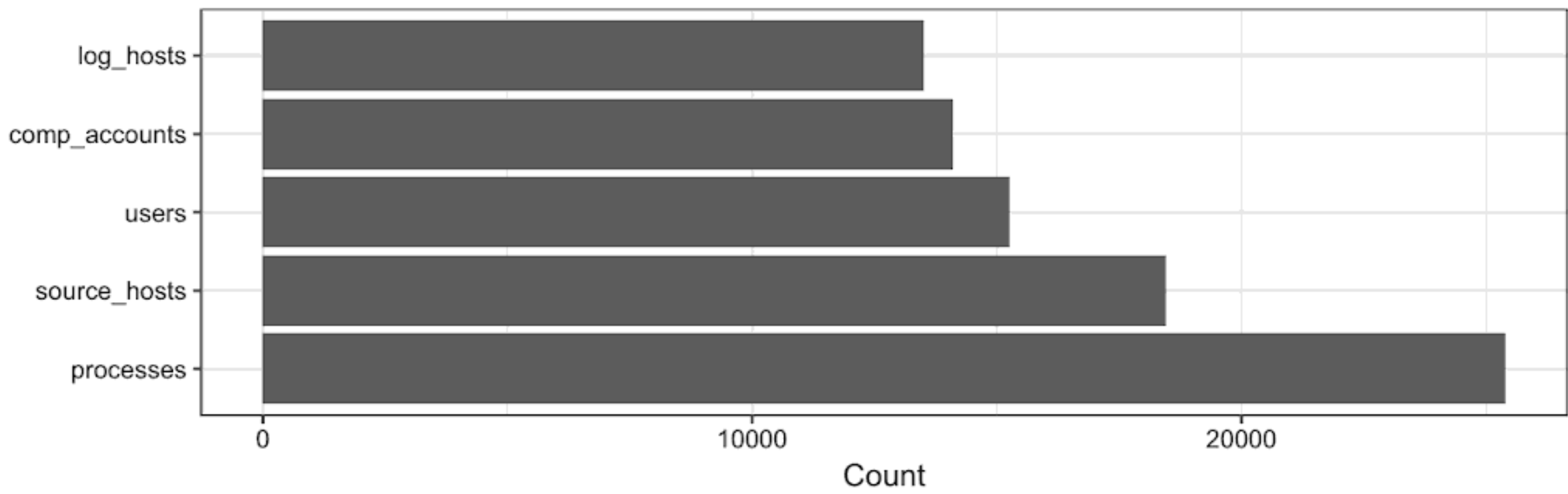


Fig. 7: Histogram of unique processes, usernames, log hosts (*LogHost*), source hosts (*Source*) and computer accounts for the whole time period.

Microsoft Active Directory domain that allow the computer to authenticate as a unique entity within the network. Figure 7 shows the count of unique processes, log hosts (*LogHost*), source hosts (*Source*), computer accounts (*UserName* ending in \$) and users (*UserName* not ending in \$) for the 90-day period. Note that the set of source hosts includes devices running non-Windows operating systems, hence there are more source hosts than log hosts. Figure 8 shows the number of wls records on a per-day basis, showing the diurnal patterns that one would expect and good collection

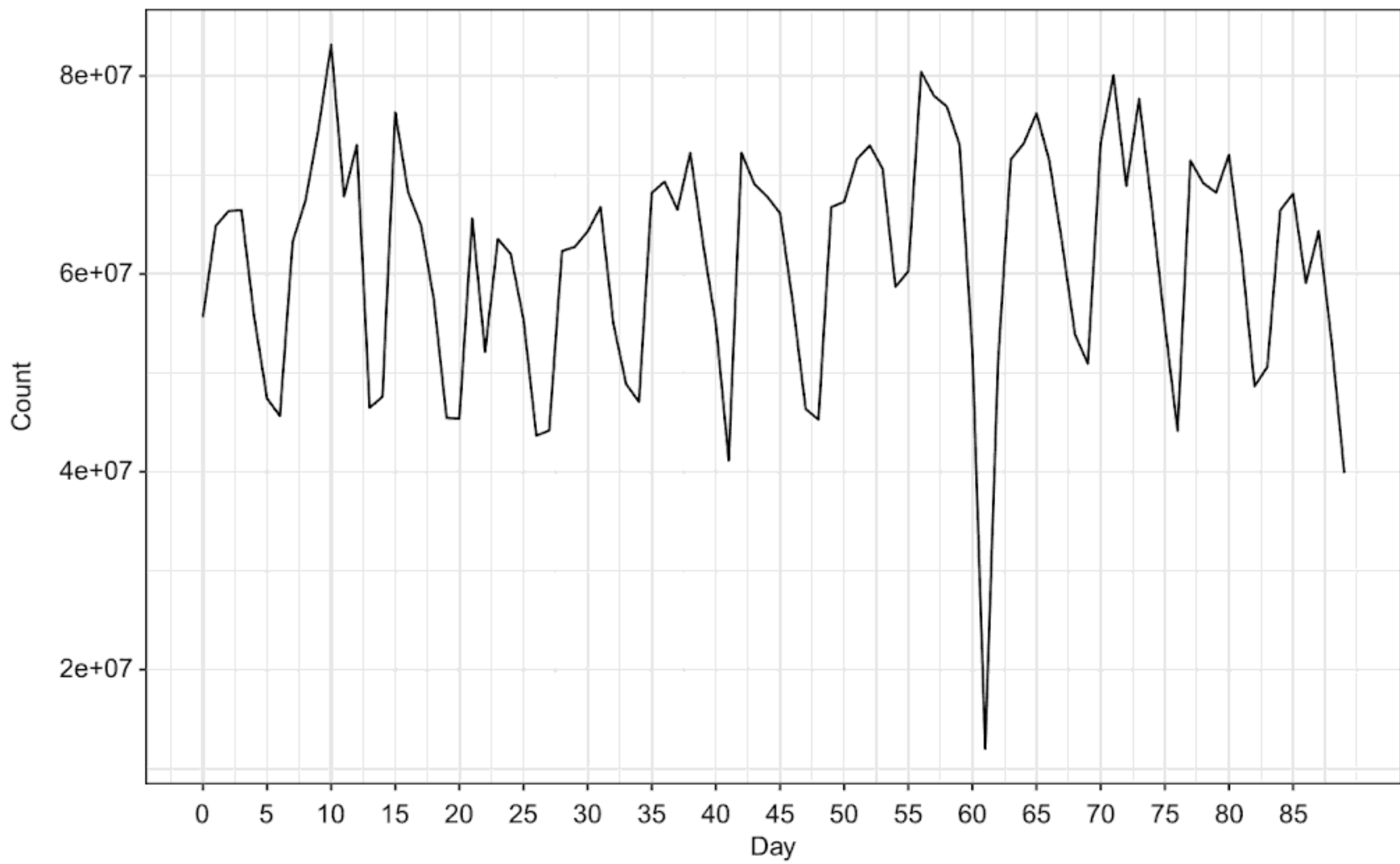


Fig. 8: Daily count of host log records.

throughout the 90 days minus a noticeable drop on day 61 similar to that of the netflow data set, Figure 1.

Requests to the Kerberos ticket granting service (TGS) (*EventID* 4769) correspond to a user requesting Kerberos authentication credentials from the Active Directory domain to a service or account name on a network computer. Hence, the *LogHost* attribute should always be an Active Directory machine and the service or account name the user is requesting access to will be given by *ServiceName*. The *ServiceName* often corresponds to a computer account on the target computer. Because this event only grants a credential, a subsequent network logon event (*EventID* 4624–*LogonType* 3) to the computer indicated by *ServiceName* is common. This differs from the previous data release (Kent, 2016), in which TGS events were assumed to be directed authentication events from the user’s machine to the computer indicated by *ServiceName*, ignoring the Kerberos intermediary.

When de-identifying the process events, only the base process name was de-identified and the extension was left as is. Further, the parent process names (*ParentProcessName*) do not have file extensions unlike the child process names (*ProcessName*); this is a direct artefact of how the process

information is logged within WLS. The missing extension can be obtained by using the *ParentProcessID* to identify the parent process start event.

Finally, many events include the *DomainName* attribute that indicates what Active Directory domain the event is associated with. The domain, combined with the *UserName*, should be considered a unique account identity. For example, user *u1* with domain *d1* is not necessarily user *u1* in domain *d2*. In addition, the domain may actually be a hostname, indicating the event does not involve a user or account associated with an Active Directory domain, but is instead a local account. Again, these accounts should be considered unique to the host indicated within the *DomainName* attribute. For example, the Administrator account on host *c1* likely does not have a relationship to the Administrator account on *c2* or the Administrator account in domain *d1*. The LANL data sets have a single primary domain, with a number of much smaller, secondary domains, and most computers have a small set of local accounts.

### 3.1. *Data parsing considerations*

While host logs can be an extremely valuable data resource for cybersecurity research, the formatting and content of the logs can vary drastically between enterprises depending upon the audit policy and technologies used to collect and forward the logs to a centralised server. Hence, parsing the data and extracting the relevant attributes is an important first step in analysing these data; see also Kent (2016).

Even though WLS provides more content and normalisation around the raw Windows logs, some challenges were still faced to provide the de-identified data.

Firstly, the semantics of attribute names are not necessarily the same for different *EventIDs* and the attribute names themselves may differ according to what tool is being used to collect and forward the logs. For example, with WLS the *UserName* for *EventID* 4774 is *MappedName*, for *EventID* 4778 and 4779 it is *AccountName* and for most other events it is *TargetUserName*. When parsing the data, these names were all standardised to *UserName*.

As with the network flow data, an extremely important task is mapping IP addresses to FQDNs. Further, unlike netflow, each record may contain both IP addresses and hostnames. The machine where the event is recorded

(*LogHost* for the de-identified data) is provided as a hostname, whereas the *Source* computer for network logons is often given as an IP address.

Finally, both usernames and process names were standardised. In some records, usernames appear with the domain name or additional characters. These discrepancies were removed from the released data in order to ensure all usernames were in canonical form. In addition, some usernames, such as “Anonymous”, “Local Service” and “Network Service”, do not map to a computer or user account. For some analyses, one may want to remove these events. In the de-identified data these commonly-seen usernames were not anonymised. For the process names, dates, version numbers, operating systems and hexadecimal strings were removed where possible so that processes run on different operating systems or with different versions would map to the same process name. For example, *flashplayerplugin\_20\_0\_0\_286.exe* would be mapped to *flashplayerplugin\_VERSION.exe*.

#### 4. Research Directions

Anomaly detection for the defensive cyber-domain is a major yet evolving research area, with much work still to be done in characterising and finding anomalies within complex cyber-data sets. Finding viable attack indicators and per computer, user and computer-to-computer models that enable anomaly detection and fingerprinting are all interesting and important research opportunities.

Although research on anomaly detection for cyber-defence spans more than two decades, operational tools are still almost exclusively rule- or signature-based. Two reasons that statistical methods have not been more widely adopted in practice are a high false-positive rate and un-interpretable alerts. Analysts are inundated with a large number of alerts and triaging them takes significant time and resources; this results in low tolerance for false alarms and alerts that provide no contextual information to guide investigation. Signature-based systems can be finely tuned to reduce false positives as they rely on very specific peculiarities that have been previously identified and documented as indicative of a cyber-attack. Further, they are interpretable as they refer to specific patterns within the data, such as weird domains, network protocols or process names.

However, despite their inherent challenges, anomaly detection methods have the advantage of being able to detect new variants of cyber-attacks and are able to keep pace with the rapidly changing cyber-attack landscape by dynamically learning patterns for normal behaviour and detecting deviations. Further, with the increasing level of encrypted network traffic, the importance of this research and the use of these methods can not be understated. Research into ways to reduce false-positives and providing interpretable anomalies will have significant impact in furthering the use of anomaly detection systems. In fact, providing interpretable anomalies can help overcome the false-positive issue as interpretability leads to quickly identifying alerts that are false positives in the same way it would enable understanding true positives. Research approaches to tackle these problems could include combining different data sets and signals, borrowing strength across entities that are similar by incorporating peer-based behaviour, community detection approaches and ways to provide meaningful context surrounding alerts to human analysts.

When using the host log data set for research, some notable characteristics of these data that need to be considered, especially if looking at the events as a time series, is periodicity and significant correlations between arrivals of different event types. This can be seen clearly in Figure 9, which shows the event times for various *EventIDs* for User205265. Periodicity in the data is often an artefact of the computer regularly renewing credentials. This explains why *EventID* 4624–*LogonType* 3 (network logon) constitutes such a significant portion of the events as seen in Figure 6. For a given entity, extrapolating higher-level, interpretable actions from the sequence of low-level events would improve modelling efforts, understanding of these data, and would itself be very useful for security analysts. See Heard *et al.* (2014) and Price-Williams *et al.* (2017) for relevant research in this area.

Another area for research with the host logs is exploring the records related to process starts and stops in detail, in particular looking at process trees. To date, little has been done in this area. Computer systems operate hierarchically; an initial root process starts many other processes, which in turn start and run descendants. A process tree is the dynamic structure that results. In theory, any process can be traced, through its ancestors, to the root process. Unusual or atypical events in process trees could indicate potential cyber-security anomalies.

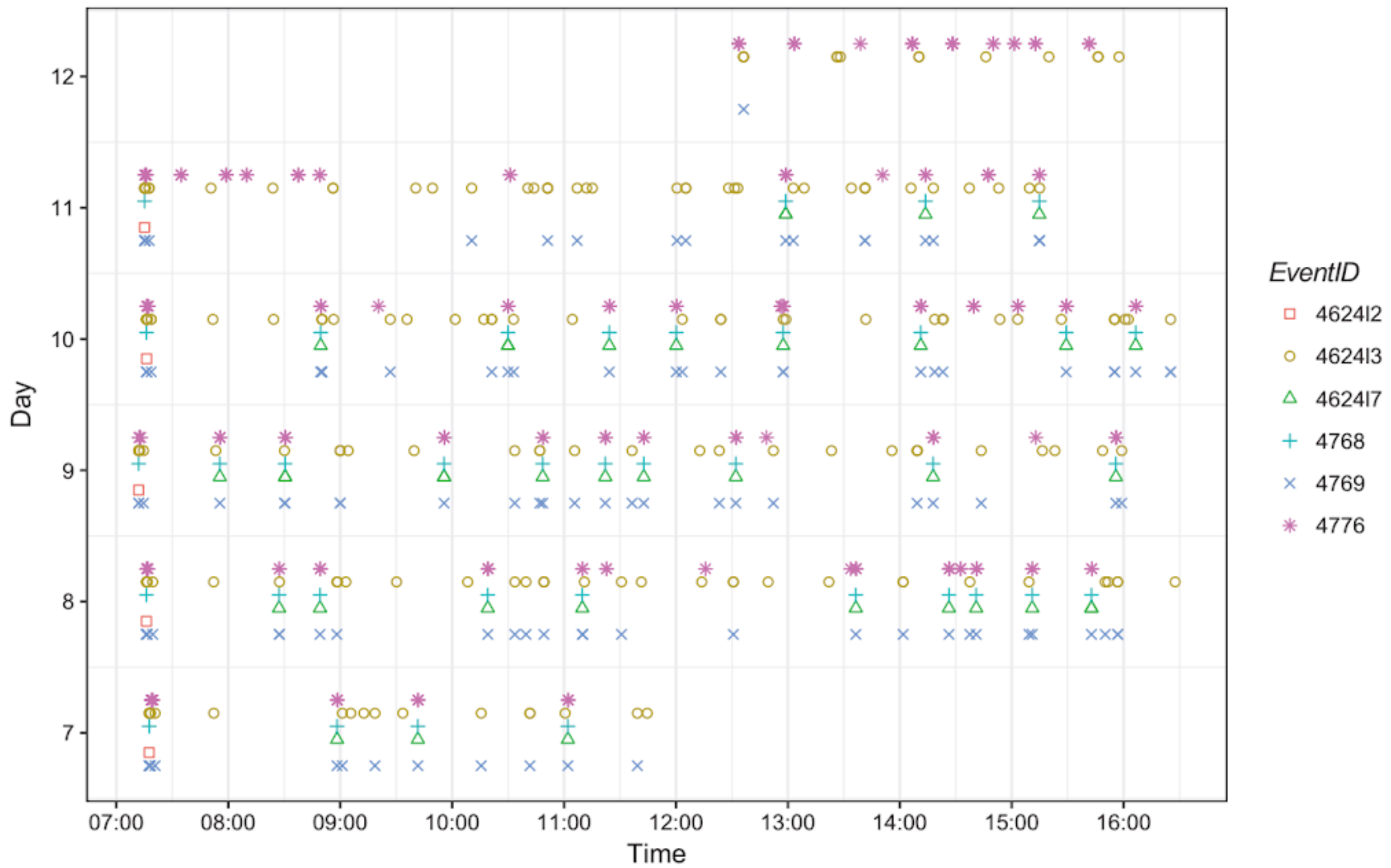


Fig. 9: Event times for User205265. 462412 corresponds to *EventID* 4624–*LogonType* 2.

Moving beyond anomaly detection, there are other important research directions for which these data could prove useful. For example, preliminary work has been done using similar data to model network segmentation and associated risk (Pope *et al.*, 2017). Using the data to build new, potential network topologies in order to reduce risk and improve security posture are viable directions. Another potential research problem is to quantify and understand data loss within cyber-data sets. The collection and normalisation processes in place for these data can result in information loss and understanding this data loss is an open problem both in general and specific to each element of the data. As most of the data elements represent people and their actions on computers, research on organisational and social behaviour is also viable using these data.

## 5. Conclusion

Operational cyber-security data sets are paramount to ensuring valuable and productive research continues to improve the state of cyber-defence. The network flow and host log event data discussed in this chapter are intended to enable such research as well as to provide an example for other potential

data set providers. In particular, while there is a considerable amount of relevant work on network data, relatively little attention has been given to host log data in the literature. Host log data are becoming increasingly relevant as endpoint security tools gain popularity within the cyber-security ecosystem. It is important that researchers embrace both the opportunity and challenge that they present. Finally, even less consideration has been given to meaningful analyses that combine these and other data sets. This paradigm shift towards a holistic approach to cyber-security defence is critical to advancing the state of the art.

## Acknowledgement

This work was supported by the US Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract DEAC52-06NA25396). The United States Government retains and the publisher, by accepting this work for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce this work, or allow others to do so for United States Government purposes.

## Appendix A. Host Log Fields

- *Time*: The epoch time of the event in seconds.
- *EventID*: Four digit integer corresponding to the event id of the record.
- *LogHost*: The hostname of the computer that the event was recorded on. In the case of directed authentication events, the *LogHost* will correspond to the computer that the authentication event is terminating at (destination computer).
- *LogonType*: Integer corresponding to the type of logon, see Table 2.
- *LogonTypeDescription*: Description of the *LogonType*, see Table 2.
- *UserName*: The user account initiating the event. If the user ends in \$, then it corresponds to a computer account for the specified computer.
- *DomainName*: Domain name of *UserName*.
- *LogonID*: A semi-unique (unique between current sessions and *LogHost*) number that identifies the logon session just initiated. Any events logged



subsequently during this logon session should report the same *LogonID* through to the logoff event.

- *SubjectUserName*: For authentication mapping events, the user account specified by this field is mapping to the user account in *UserName*.
- *SubjectDomainName*: Domain name of *SubjectUserName*.
- *SubjectLogonID*: See *LogonID*.
- *Status*: Status of the authentication request. “0 × 0” means success otherwise failure; failure codes for the appropriate *EventID* are available online.<sup>f</sup>
- *Source*: For authentication events, this will correspond to the the computer where the authentication originated (source computer), if it is a local logon event then this will be the same as the *LogHost*.
- *ServiceName*: The account name of the computer or service the user is requesting the ticket for.
- *Destination*: This is the server the mapped credential is accessing. This may indicate the local computer when starting another process with new account credentials on a local computer.
- *AuthenticationPackage*: The type of authentication occurring including Negotiate, Kerberos, NTLM plus a few more.
- *FailureReason*: The reason for a failed logon.
- *ProcessName*: The process executable name, for authentication events this is the process that processed the authentication event. *ProcessNames* may include the file type extensions (i.e., exe).
- *ProcessID*: A semi-unique (unique between currently running processes AND *LogHost*) value that identifies the process. *ProcessID* allows you to correlate other events logged in association with the same process through to the process end.
- *ParentProcessName*: The process executable that started the new process. *ParentProcessNames* often do not have file extensions like *ProcessName* but can be compared by removing file extensions from the name.
- *ParentProcessID*: Identifies the exact process that started the new process. Look for a preceding event 4688 with a *ProcessID* that matches this *ParentProcessID*.

---

<sup>f</sup><https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx>.

## Appendix B

Table B.1: Event attributes.

<i>EventIDs</i>	<i>Attribute</i>
All	<i>Time</i>
All	<i>EventID</i>
All	<i>LogHost</i>
4624, 4625, 4634	<i>LogonType</i>
4624, 4625, 4634	<i>LogonTypeDescription</i>
All except System Events	<i>UserName</i>
All except System Events	<i>DomainName</i>
All except 4768, 4769, 4770, 4774, 4776	<i>LogonID</i>
4624 ( <i>LogonType</i> 9), 4648, 4774	<i>SubjectUserName</i>
4624 ( <i>LogonType</i> 9), 4648, 4774	<i>SubjectDomainName</i>
4624 ( <i>LogonType</i> 9), 4648	<i>SubjectLogonID</i>
4768, 4769, 4776	<i>Status</i>
4624, 4625, 4648, 4768, 4769, 4770, 4776	<i>Source</i>
4769, 4770	<i>ServiceName</i>
4648	<i>Destination</i>
4624, 4625, 4776	<i>AuthenticationPackage</i>
4625	<i>FailureReason</i>
4624, 4625, 4648, 4688, 4689	<i>ProcessName</i>
4624, 4625, 4648, 4688, 4689	<i>ProcessID</i>
4688	<i>ParentProcessName</i>
4688	<i>ParentProcessID</i>

## References

- Barbosa, R. R. R. (2014). *Anomaly Detection in SCADA Systems — A Network Based Approach*, Ph.D. thesis, Centre for Telematics and Information Technology, University of Twente, Netherlands.
- Berthier, R., Cukier, M., Hiltunen, M., Kormann, D., Vesonder, G. and Sheleheda, D. (2010). Nfsight: Netflow-based network awareness tool, in *Proceedings of LISA10: 24th Large Installation System Administration Conference*, p. 119.
- Claise, B. (2004). Cisco systems NetFlow services export, Version 9, RFC 3954, Internet Engineering Task Force.
- Cyber Systems and Technology Group (1998). DARPA intrusion detection data sets, URL: <https://www.ll.mit.edu/ideval/data/>.
- Glasser, J. and Lindauer, B. (2013). Bridging the gap: A pragmatic approach to generating insider threat data, *2012 IEEE Symposium on Security and Privacy Workshops*, pp. 98–104.

- Heard, N., Rubin-Delanchy, P. and Lawson, D. J. (2014). Filtering automated polling traffic in computer network flow data, in *2014 IEEE Joint Intelligence and Security Informatics Conference*, pp. 268–271.
- Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. and Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX, *IEEE Communications Surveys & Tutorials* **16**, 4, pp. 2037–2064.
- Kent, A. D. (2014). User-computer authentication associations in time, Los Alamos National Laboratory, doi:10.11578/1160076.
- Kent, A. D. (2016). Cyber security data sources for dynamic network research, in N. Adams and N. Heard. eds., *Dynamic Networks and Cyber-Security*, Vol. 1, p. 37, World Scientific, UK.
- Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2009). Identifying suspicious URLs: An application of large-scale online learning, in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 681–688.
- Minarik, P., Vykopal, J. and Krmicek, V. (2009). Improving host profiling with bidirectional flows, in *International Conference on Computational Science and Engineering, 2009. CSE'09.*, Vol. 3, pp. 231–237.
- Nguyen, K. V., Tyagi, N. K. and Lau, R. M. (2017). Flow de-duplication for network monitoring, US Patent 9,548,908.
- Pope, A., Tauritz, D. and Kent, A. (2017). Evolving bipartite authentication graph partitions, *IEEE Transactions on Dependable and Secure Computing*, **99**, pp. 1–1.
- Price-Williams, M., Heard, N. and Turcotte, M. (2017). Detecting periodic subsequences in cyber security data, in *IEEE European Intelligence and Security Informatics Conference (EISIC2017)*, pp. 84–90.
- Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A. and Stiller, B. (2010). An overview of IP flow-based intrusion detection, *IEEE Communications Surveys and Tutorials* **12**, 3, pp. 343–356.
- Trammell, B. and Boschi, E. (2008). Bidirectional flow export using IP Flow Information Export (IPFIX), RFC 5103, Internet Engineering Task Force.

## Chapter 2

# Computational Statistics and Mathematics for Cyber-Security

**David J. Marchette**

*Naval Surface Warfare Center,*

*Dahlgren, VA 22448, USA*

*david.marchette@navy.mil*

Computer and network security relies on many different tools, such as secure programming practices, firewalls, virus scanners and various algorithms to detect attacks and malicious software. The latter require the analysis of complex and varied data such as packet streams, emails, potential malicious binary executable files and user activity on a computer and on the network. Modern data analytics has a number of tools to analyse these data streams and to design detection algorithms. This chapter discusses several such tools that come from the computational statistics literature and from pure mathematics: nonparametric probability density estimation, graph-based manifold learning, topological data analysis. These ideas are illustrated on a problem of malware classification and on network data.

### 1. Introduction

In an influential paper in statistical science (Breiman, 2001b), Leo Breiman described two cultures of data analysis, which Donoho (2015) refers to as prediction and inference. The idea is that there are those who focus on the bottom line — how well does the algorithm perform on a given task of interest, such as classification or regression — and those who are more concerned with making inference, such as using a model, and the parameters of the model fitted from data, to make statements about the world (or some small piece of it of particular interest to the analyst). Of course, these are not mutually exclusive, and we all (one hopes) have at least some of both of these in our make-up; however, there is a distinct difference in these two

cultures. One way of making the distinction, which Donoho discusses at some length, is between those who are primarily interested in elucidating the model that generated the data, and those interested simply in how well the model predicts (some aspect of) future observations.

In cyber-security, one is mostly interested in the latter. While there is considerable interest in things like packet arrival times and other statistics of networks, these are more important for the design and implementations of networks rather than for the security aspects. From a security perspective one mostly cares about whether one can protect the network, computer and data. The focus is more on whether attacks and intrusions can be detected rather than in modelling these attacks. With that said, it is important to note that often one is reduced to modelling what is normal, or at least benign, and looking for deviations from this model, and so one must not completely remove oneself from the inference camp.

In this chapter, I will discuss several statistical methods for analysing, modelling, and predicting in complex data that are of interest for applications in computer security. I will use for illustration a data set provided by Kaggle.<sup>a</sup> The data consist of 10,868 examples of malware, grouped into nine malware families. The task is to construct a classifier that can determine the family of new malware. The data comes in two forms: a byte-dump of the executable programme, and a decompilation of the programme into assembly code. In both cases, some adjustments have been made so that an executable version of the programme cannot be generated by a simple manipulation of the data. I will use the byte-dump data from this data set.

In addition to this data set, I will also utilise the network data available from the Los Alamos National Laboratory (LANL) (Kent, 2016).<sup>b</sup> There are several interesting data sets there, and I will use the network flows data.

The layout of the chapter is as follows. First, I will consider some basic applications of computational statistics in Section 2. In Section 3, I will consider some dimensionality reduction techniques that allow one to go from complex, high-dimensional observations to lower-dimensional data which (one hopes) is easier to model, without losing “too much” of the information in the data. Finally, Section 4 will discuss some new methods

---

<sup>a</sup><https://www.kaggle.com/c/malware-classification>.

<sup>b</sup>Data available at <http://csr.lanl.gov/data/cyber1/>.

derived from algebraic topology that have potential for becoming a new generation of data analysis tools.

## 2. Computational Statistics

Computational statistics is generally described as the interface between statistics or data analysis and computing.<sup>c</sup> It is the collection of tools and theory developed to model large, high-dimensional and complex data, and the algorithms and computer code that implement these ideas. Generally the tools are nonparametric rather than parametric, and the data of interest are either high dimensional (more than three or four variables), consist of a large number of observations (more than a few hundred), complex (a mix of continuous and categorical variables, or other data types) or a combination of the above. Cyber-security data has all of these properties, and in many cases the data volumes are truly massive. Thus, computationally efficient methods are essential for the analysis of the data. In this section, I will look at a few common methods from computational statistics related to probability density estimation, classification and machine learning.

### 2.1. Density estimation

First, let's focus on univariate data. Everyone is familiar with the histogram, which can be thought of as simply counting the number (proportion) of observations that fall in each of a set of bins.<sup>d</sup> In one extreme case, where the data are discrete, the bins can correspond to the unique values of the data, and the histogram is simply the empirical estimate of the probability mass function. In the continuous case, one generally takes the range of the data and selects equal-sized bins that cover this range. For example, if the data fall in the interval  $[0, 1]$  and one wants  $m$  bins, the bins can be  $[0, \frac{1}{m}]$ ,  $(\frac{1}{m}, \frac{2}{m}]$ ,  $\dots$ ,  $(\frac{m-1}{m}, 1]$ . It is well known (and often not emphasised enough in school) that the shape of the histogram (the estimate of the probability density function) can vary quite a bit by changing the placement of the bins. For example, instead of placing the center of the first bin at  $\frac{1}{2m}$ , placing it at the smallest observation will result in a different "picture" and,

---

<sup>c</sup>[https://en.wikipedia.org/wiki/Computational\\_statistics](https://en.wikipedia.org/wiki/Computational_statistics).

<sup>d</sup>For this discussion, the histogram consists of equal-sized adjacent bins.

more importantly, a different estimate of the density.<sup>e</sup> Many solutions have been proposed to solve this problem (David Scott’s book (Scott, 1992) is an excellent reference). The one I will consider is kernel density estimation (Silverman, 1986). If one thinks of the histogram as “place the bins, then count the points”, the kernel density estimate corresponds to “place the points and count the bins”. Formally

$$f_{ke}(x) = \frac{1}{nh} \sum_{i=1}^n K \left( \frac{x - x_i}{h} \right).$$

Here,  $K$ , the kernel, is a probability density function, such as the normal (Gaussian) distribution.<sup>f</sup> The idea is that one places the “kernel” at every point, and then adds up the contribution from all of the kernels.

Note that the kernel estimator, like the histogram, has a single parameter,<sup>g</sup>  $h$ , called the bandwidth.<sup>h</sup> Larger values of  $h$  give a smoother estimate of the density, smaller values produce less-smooth estimates, and can detect fine structure in the density. Note that, like the number of bins in a histogram, the bin width should be chosen as a function of  $n$ , with larger  $n$  corresponding to smaller bandwidths. There are many rules of thumb for bandwidth selection (Silverman, 1986; Wand and Jones, 1994). An important fact about kernel estimators is that they are “consistent”, in the sense that (with weak assumptions) the estimate converges to the true density so long as the bandwidth decreases in  $n$ , the number of observations, at an appropriate rate. Asymptotic results of this type may not be directly useful in a practical application,<sup>i</sup> but they do provide some confidence that as long as one has “enough” data, one will obtain a “good” estimate, and if this

---

<sup>e</sup>Changing the number of bins will also change the density estimate, and this will be illustrated in discussion of the kernel estimator.

<sup>f</sup>Technically,  $K$  needs to be positive, integrate to one, and have finite variance — be square-integrable — but some of these conditions can be weakened.

<sup>g</sup>Technically, as mentioned above, the histogram has a second parameter, the position of the first bin. Since this is almost always set according to the minimum value of the data, or *a priori* by knowledge of the possible data range, it is reasonable to think of the histogram as being parametrised by the bin width.

<sup>h</sup>The bandwidth corresponds to the bin width (or equivalently the number of bins, assuming adjacent bins) of the histogram.

<sup>i</sup>The curse of dimensionality means that for high-dimensional data, asymptotic results are particularly unhelpful.

estimate performs well on the prediction task, there is reason to believe it is likely to perform well on new data.

Although there are good reasons to select the kernel  $K$  according to some knowledge of the data, I will use Gaussian (normal density) kernels in this work. The reader is encouraged to look at the discussions about kernels in the references.

The kernel estimator can be seen as an extreme case of a mixture model (McLachlan and Peel, 2000): one fits a mixture of a given density to the data. For example, the Gaussian mixture model (with  $m$  terms) is

$$f_{gmm}(x) = \sum_{j=1}^m \pi_j \phi(x; \mu_j, \sigma_j).$$

The mixture proportions  $\pi_i$  are positive and sum to one. There are many variations on this theme (Fraley and Raftery, 2002; Marchette *et al.*, 1996; Priebe, 1994). Fitting the parameters of the mixture can be tricky, but there are good methods using variations of the expectation-maximisation (EM) algorithm (McLachlan and Krishnan, 1997) and clever computer science (Moore, 1998). In text data analysis, these show up (using different densities and usually a Bayesian approach to the parameters) as topic models (Blei and Lafferty, 2009; Blei *et al.*, 2003).

Consider the distribution of bytes in malware. That is, for each value from 0 to 255, one computes the proportion of times each byte is found in the file.<sup>j</sup> This seems (and is) a pretty naive set of features to use, but I'll use it to illustrate some of these ideas.

Figure 1 shows the empirical probability distribution (the proportions of times each byte occurs in the file) for one of the pieces of malware. The curve is the kernel estimator. The graph has been scaled — the proportion of 0 bytes for this code is 0.14, which accounts for the vertical offset of the kernel estimator — it is smoothing out this value. It is clear, though, that some of the structure of the data is represented in the kernel estimator. Further, the structure is not simply a random mess — there are what appear to be clusters of similar bytes in the data.

There are multivariate versions of kernel estimators and mixture models. These essentially replace the kernel or mixture component with a

---

<sup>j</sup>In the data provided by Kaggle, there are bytes with value “??”, corresponding to obfuscation to prevent the reconstruction of an executable from the data. In these studies, I remove these values.



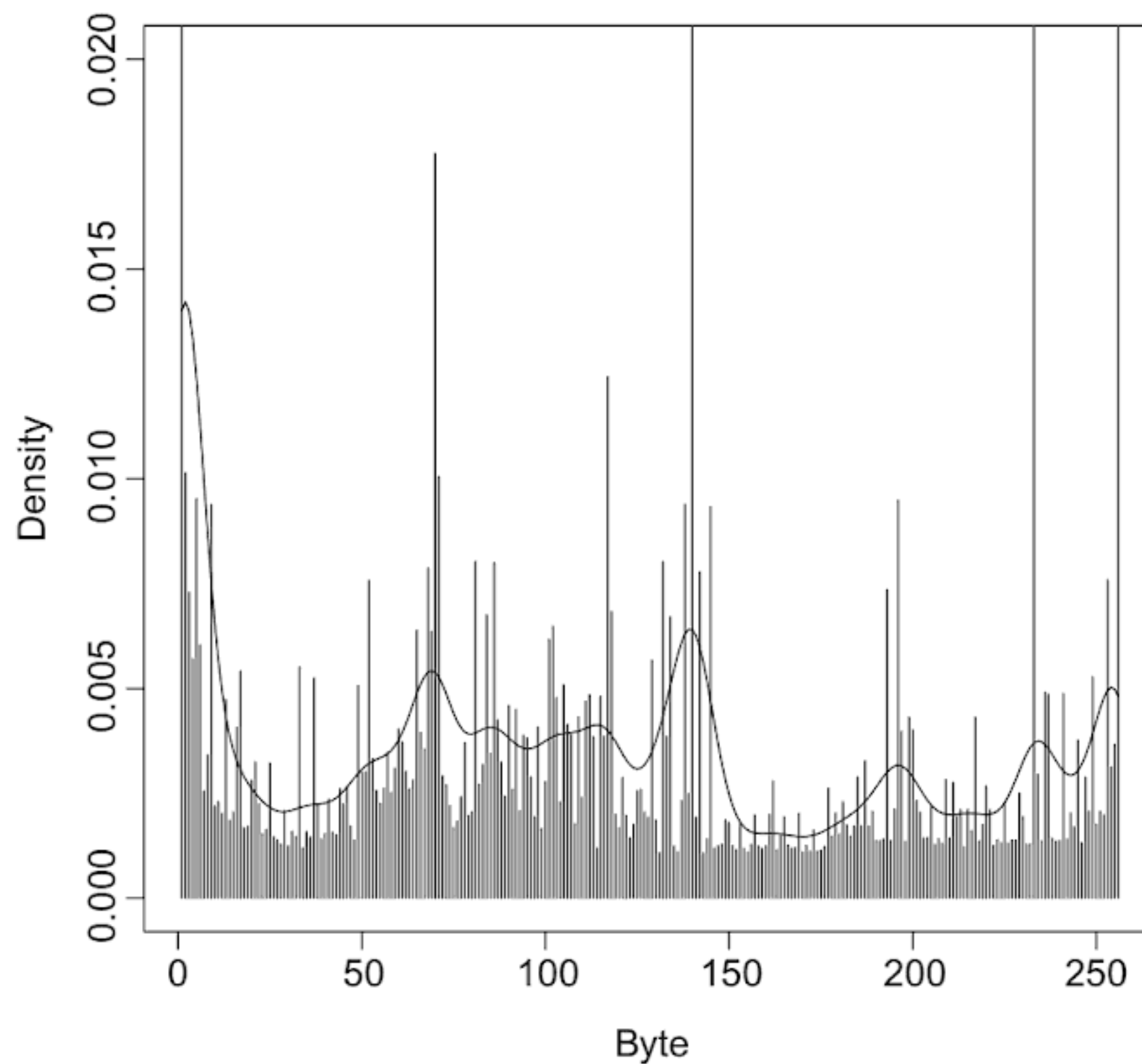


Fig. 1: A one-dimensional kernel estimator of one of the malware programmes. The bars correspond to the empirical estimate of the probability mass function (the histogram with bins centred on the integers). The y-axis has been scaled to show the fine-scale structure.

multivariate kernel or density. Details can be found in the references. See Figure 2 for an example computed on a single malware programme. Here, instead of counting the number of times a byte occurs, we are counting the number of times a pair of adjacent bytes occur.

Now let's investigate the utility of the byte-counts for a simple task: classify the malware according to family. For this, and other tests, I have selected (at random) 100 observations from each class (except for class 5 which only has 42 observations — I select 21 of them) to use as training, and use the rest of the observations for testing. All errors reported in this paper are computed on the testing data.

Using the histogram,<sup>k</sup> a nearest neighbour classifier<sup>l</sup> has an error of 0.162. The kernel estimate of the density function (treating the data as if they were continuous) results in a nearest neighbour error of 0.124. The McNemar test (Agresti, 1990), which is a Chi-square test for symmetry of

<sup>k</sup>In this case, the histogram is simply the byte-counts (proportions) for the 256 possible byte values and the bin size is 1. This is the empirical estimate of the probability mass function.

<sup>l</sup>The use of a nearest neighbour classifier is for the purposes of comparison only. As is well known, this classifier is rarely a viable choice for real-world problems, but it can be instructive for getting a feel for the data, and for upper bounds on the error one can expect for the problem.

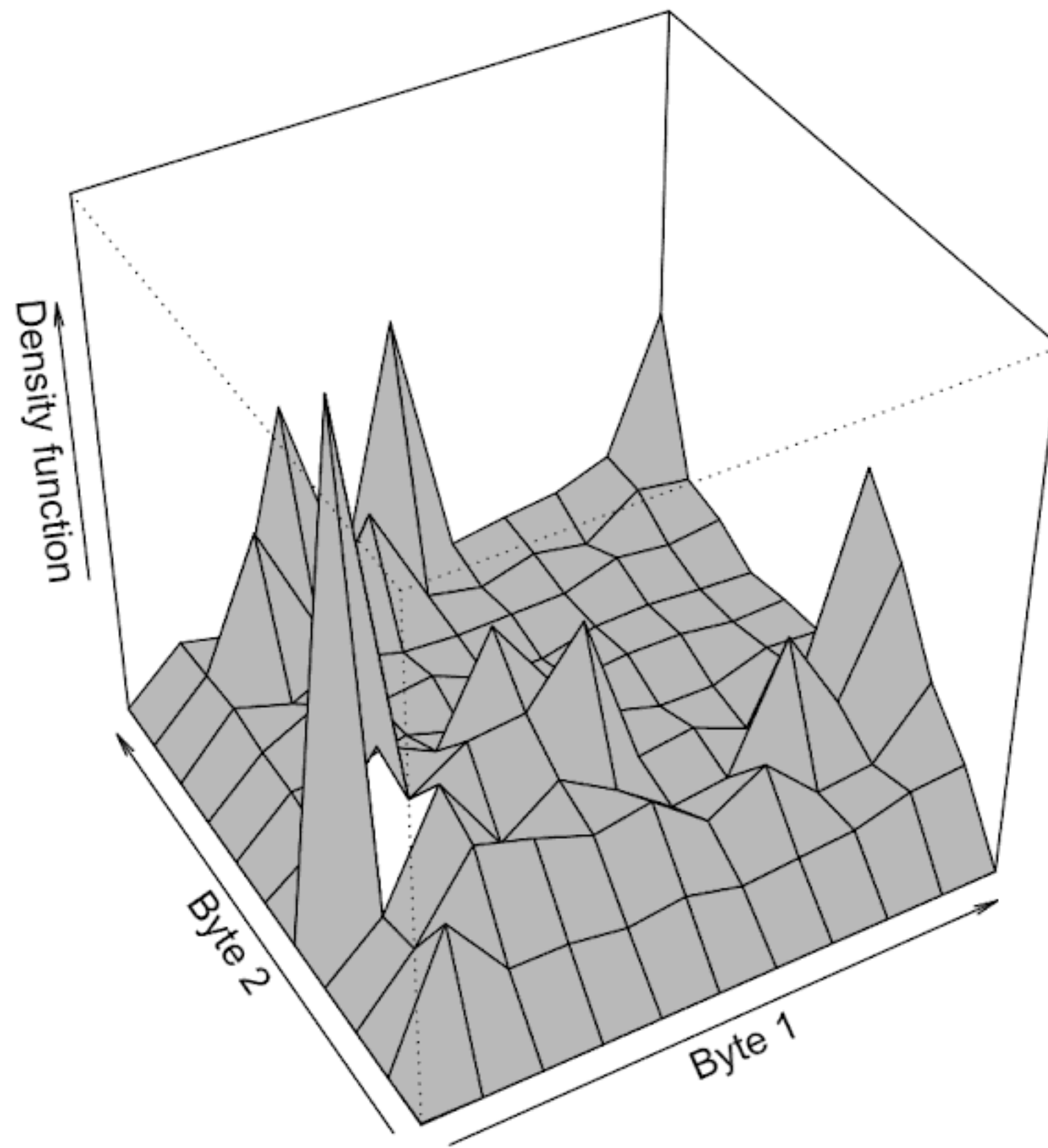


Fig. 2: A two-dimensional kernel estimator of one of the malware programmes. Adjacent bytes correspond to the two axes. This has been smoothed to a  $32 \times 32$  grid.

Table 1: Confusion matrix for the nearest neighbour classifier on byte-counts for the nine class malware data. Blank entries are 0s.

		True Class						
1291	116		6	2	53	12	75	128
5	1525		1		4	1	6	5
0	33	2807	3		1	2	2	0
1	63	29	352		12	2	7	10
27	89			19	13	8	23	21
55	166	2	10		554	2	32	28
30	6	4	3		4	244	45	0
28	243				3	15	920	12
4	137				7	12	18	709

the contingency table (whether the two classifiers miss the same number of observations) has a  $p$ -value essentially 0, and so the kernel estimator on these data is a better classifier (see Tables 1 and 2). Note that although the kernel estimator produces better performance overall, it is not better for all classes.

Figure 3 shows how the individual bytes are correlated with class membership, a crude indicator of their importance for classification. Byte value

Table 2: Confusion matrix for the nearest neighbour classifier on a kernel estimator from byte-counts for the nine class malware data.

		True Class						
1232	133	1	4	17	9	65	64	
16	1816	2		5	6	6	26	
0	13	2811	2	1		3	3	
4	31	15	363		14	6	7	
34	4	1	2	15	9		19	
52	57	8	4	1	581	2	17	
7	30	4			1	271	24	
43	74	3		1	17	7	957	
53	220		1		6	3	31	
							757	

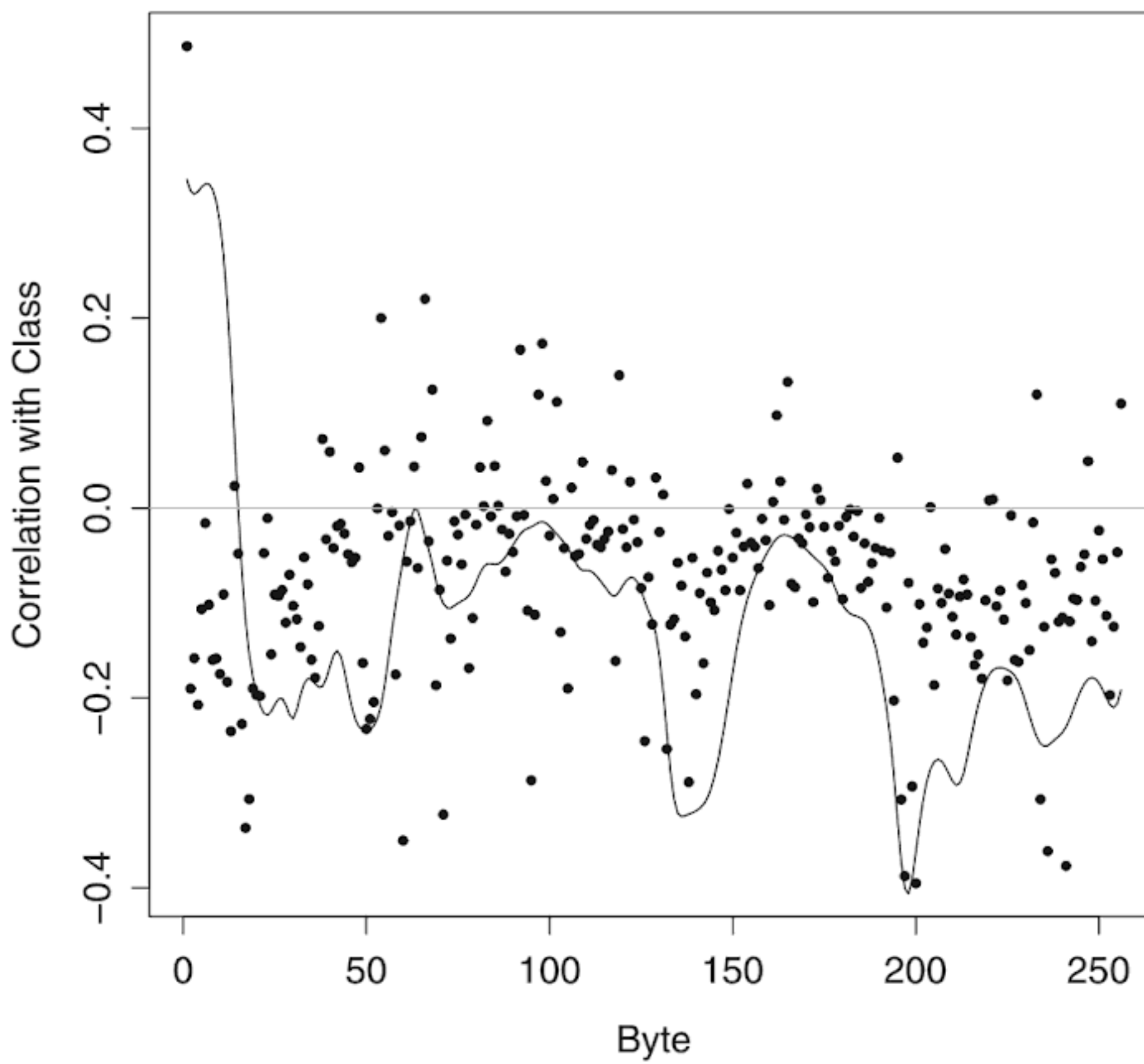


Fig. 3: Correlation with class for the byte proportions (dots) and the kernel estimator (curve).

0 is the most correlated, with values 198, 201, 237 and 242 being the most negatively correlated. The kernel estimator, which smooths the histogram, also smooths this correlation and suggests that the last two may not be as important. Note also that the kernel estimator has a very different profile of correlation.

Criticisms of this little experiment are easy to come by, but I want to make two points. First, simple ideas (byte-counts, nearest neighbour classifiers) can often provide surprising performance and can be used to gain further insight into the problem. Second, statistical ideas such as smoothing (kernel density estimation), variable selection and dimensionality reduction (which we'll see below) and modelling can improve performance dramatically even in the case where domain knowledge does not provide much help.<sup>m</sup>

If one uses only the first 1000 bytes in the file, the results swap: in this case, the histogram obtains an error of 0.130 and the kernel estimator an error of 0.176. Note that the kernel estimate on the entire file is still the superior classifier. Domain knowledge can be used to determine what the optimal approach is: Is it important to process the entire file, or are there particular sections (such as the beginning) that are more informative? Should different sections be weighted differently?

It is important to note that the bytes represent a fairly complicated type of data. On the one hand, they are categorical, representing (one byte of) a machine instruction code. On the other hand, some of them represent memory locations or data, in which case they are integers, and also may correspond to (one byte of) floating point numbers. If the data were purely categorical, the kernel estimator approach above would be nonsensical, and (presumably) this would be reflected in the performance.

Another important consideration when using kernel estimators is to account for a known (or suspected) constraint on the range. Since we know there are no values outside of  $[0, 255]$ , it makes sense to use a variant of the kernel estimator that constrains the estimate to be within this range. I didn't bother, since the ultimate purpose is not to produce the best estimate of the probability density possible, but to produce the best performance of a classifier.<sup>n</sup>

---

<sup>m</sup>Admittedly, in this case I chose to use no domain knowledge, to illustrate the idea. At a minimum, knowing (and utilising) the word size should be a first step when processing data corresponding to machine instructions.

<sup>n</sup>Note that the Bayes error is given by the classifier that uses the true joint probability distribution of the data, *not* the probability distribution of individual byte-counts. So, it is not the case that a better estimate will necessarily produce a better classifier.

It is important to note that the above is not an estimate of the probability density function of the (256-dimensional) data. Instead, I am using density estimates *as the features* which are then passed to the machine learning algorithm. This is in the spirit of much of the text processing literature (Aggarwal and Zhai, 2012; Gupta and Lehal, 2009), where one often starts with term-frequencies computed on a per-document basis.

## 2.2. Streaming data

The malware problem discussed in the previous section has the property that it comes in discrete chunks (single files) at a relatively slow pace. Internet packet traffic is an example of streaming data: there is an extremely high volume of data that comes at a very rapid pace, making it difficult to store and process all the data in the manner described above. Streaming, or “on-line” methods are needed to allow the processing of the data.

To illustrate, consider the problem of estimating the mean of a univariate random variable  $X$ . One observes data in time,  $\{x_1, x_2, \dots\}$  and can only store a small number of bytes. We’d like to know, at any given time, what the sample mean is for all the data seen to date. Fortunately, there is a simple recursive formula for the sample mean that only requires the storage of the current estimate and the number of observations seen to date:

$$\widehat{X}_n = \frac{n-1}{n} \widehat{X}_{n-1} + \frac{1}{n} X_n. \quad (1)$$

Note that (1) is exact: after  $n$  observations,  $\widehat{X}_n$  is the sample mean of the data  $\{x_1, x_2, \dots, x_n\}$ .

One can implement an exponential window on the data, “forgetting” past data, which can be useful in a nonstationary environment, by making a simple change to the equation:

$$\widehat{X}_n = \frac{N-1}{N} \widehat{X}_{n-1} + \frac{1}{N} X_n. \quad (2)$$

Here,  $N$  is fixed, with its value controlling the width of the exponential window.

There are many approaches to analysing streaming data in the literature. Various streaming density estimates have been investigated, including kernel estimators (Wegman and Marchette, 2003), wavelet-based density estimators (Caudle *et al.*, 2015) and mixture models (Priebe, 1994). The

first two citations were applied to network data, and are thus of particular interest to the computer security community.

The basic idea of most streaming algorithms comes down to a rewriting of (2). For example, a streaming density estimator (Wegman and Marchette, 2003) can be defined as

$$\hat{f}_n(x) = \theta \hat{f}_{n-1}(x) + (1 - \theta)\gamma(x, X_n). \quad (3)$$

Here,  $\gamma(\cdot)$  is whatever is appropriate for a single observation (such as the contribution of the observations to a kernel estimator, or an update to the parameters of a mixture model). It is generally straightforward to modify (3) to allow a small number of recent observations to be used, rather than just a single observation, provided the data rates and memory overhead allows for this.

In particular, consider

$$\hat{f}_n(x) = \theta \hat{f}_{n-1}(x) + (1 - \theta)\frac{1}{h}K\left(\frac{x - x_n}{h}\right). \quad (4)$$

This presupposes that one knows the values at which one is going to want to apply the estimator: one needs to keep track of (4) at each point at which one wishes to evaluate the kernel estimator. For the malware data of byte-counts, we only have a finite number of possible values. Otherwise one would want to assign a grid to the possible values.

The flows data from Los Alamos consists of number of packets and bytes for a collection of flows between computers over time. Figure 4 depicts the number of packets and bytes for a single day. Note the obvious lines in this plot, indicating flows that have very similar payloads per packet — this is an indication of the type of application that is associated with the flow. The linear structure indicates that a polar coordinates representation is appropriate for these data, and this is depicted in Figure 5. In both cases I have used alpha-blending to reduce the overplotting. Thus, dark regions are regions with many points overplotted within the region.

There is a lot of structure evident in these figures. Diagonal lines in Figure 4 and vertical lines in Figure 5 are indicative of applications with the same ratio of bytes to packets — presumably applications whose flows consist of many same sized packets.

To illustrate the streaming kernel estimator, consider Figure 6. Here, we are estimating the probability density of the log of the number of bytes

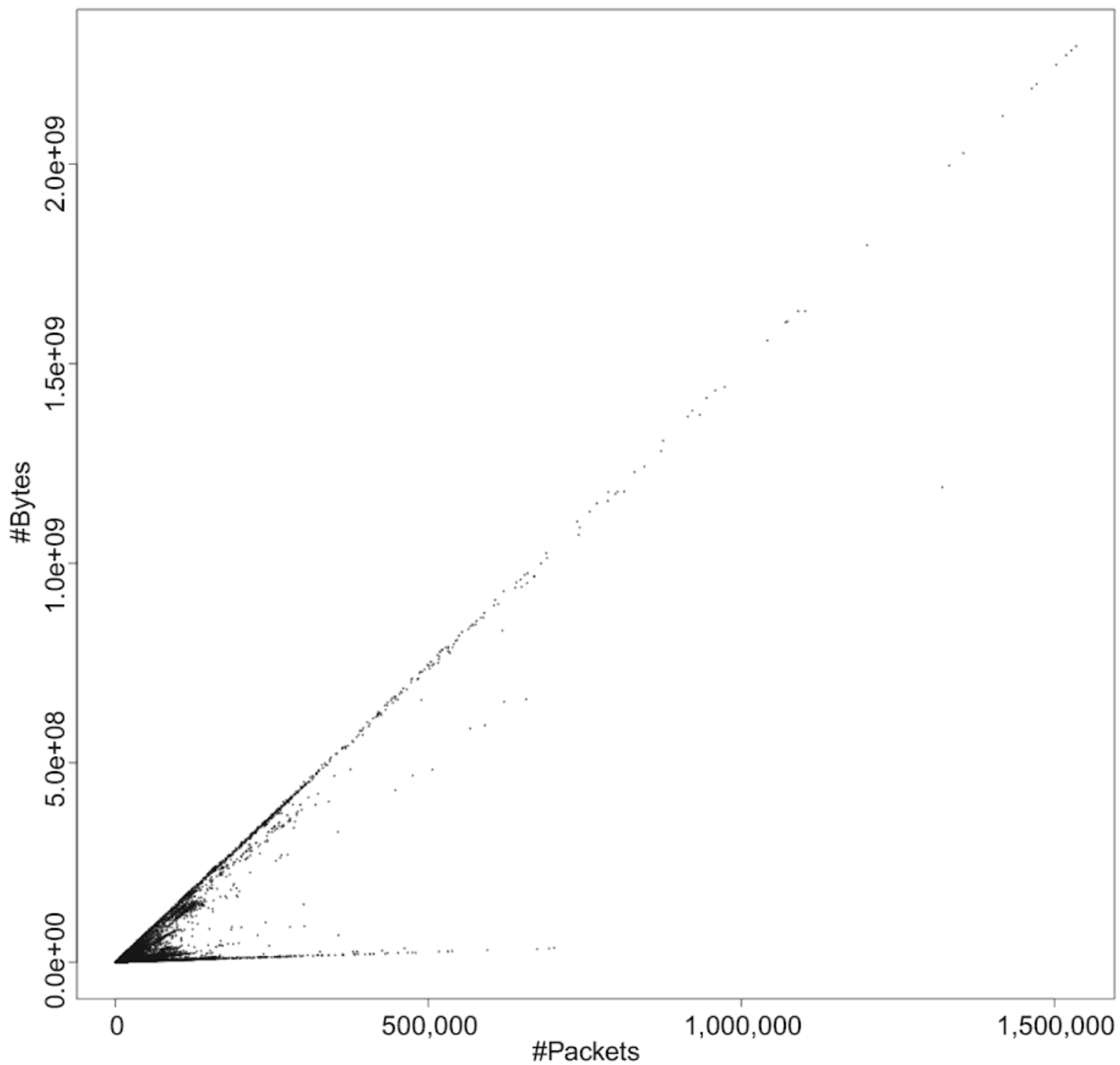


Fig. 4: Number of packets against the number of bytes for one day's worth of data.

in a web session: ports 80, 8080 or 443. We start with an estimate on the first 1000 sessions, and then update the estimate using (4) with each new session. This figure corresponds to 9 hours of data.

There is obvious temporal structure in the data; however, the time period is too short to see if the periodic structures that are apparent in the figure are continued. There is also some structure in the sizes (number of bytes) of the web sessions. Clear peaks are evident, and persist through time. This shows that there is some stability in the application, which is probably due to the way web interactions work; the text of a web page may be transferred via one session while the (substantially larger) images on the page are transferred via other sessions. There is also some standardisation in image size, in the sense that these are often designed to fit in a standard window, and often use the same type of compression (JPEG) throughout.

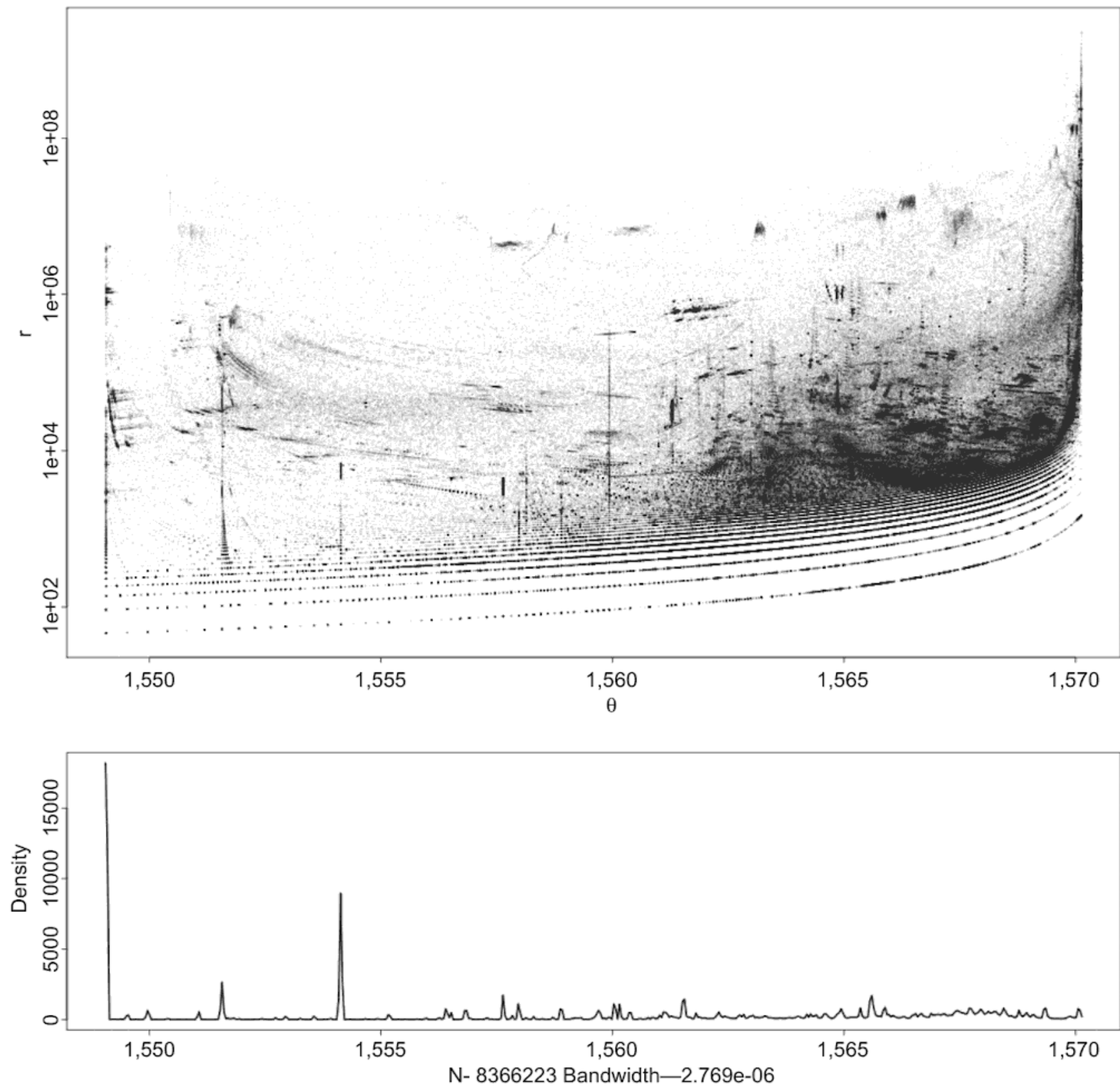


Fig. 5: Polar coordinates of the flows data (top) and a kernel density estimator of the angle  $\theta$  (bottom). This corresponds to a single day's worth of data.

This may account for the fairly consistent peaks in the higher ranges of the  $x$ -axis.

### 2.3. Machine learning

The field of machine learning deals with the problems of developing algorithms to allow the computer to “learn” patterns from data. For the purposes of cyber-data analytics, one wants to perform one of three basic tasks: classification (also called supervised learning), where the computer is given labelled data and the task is to learn how to correctly label new observations; clustering (also called unsupervised learning), where the data are



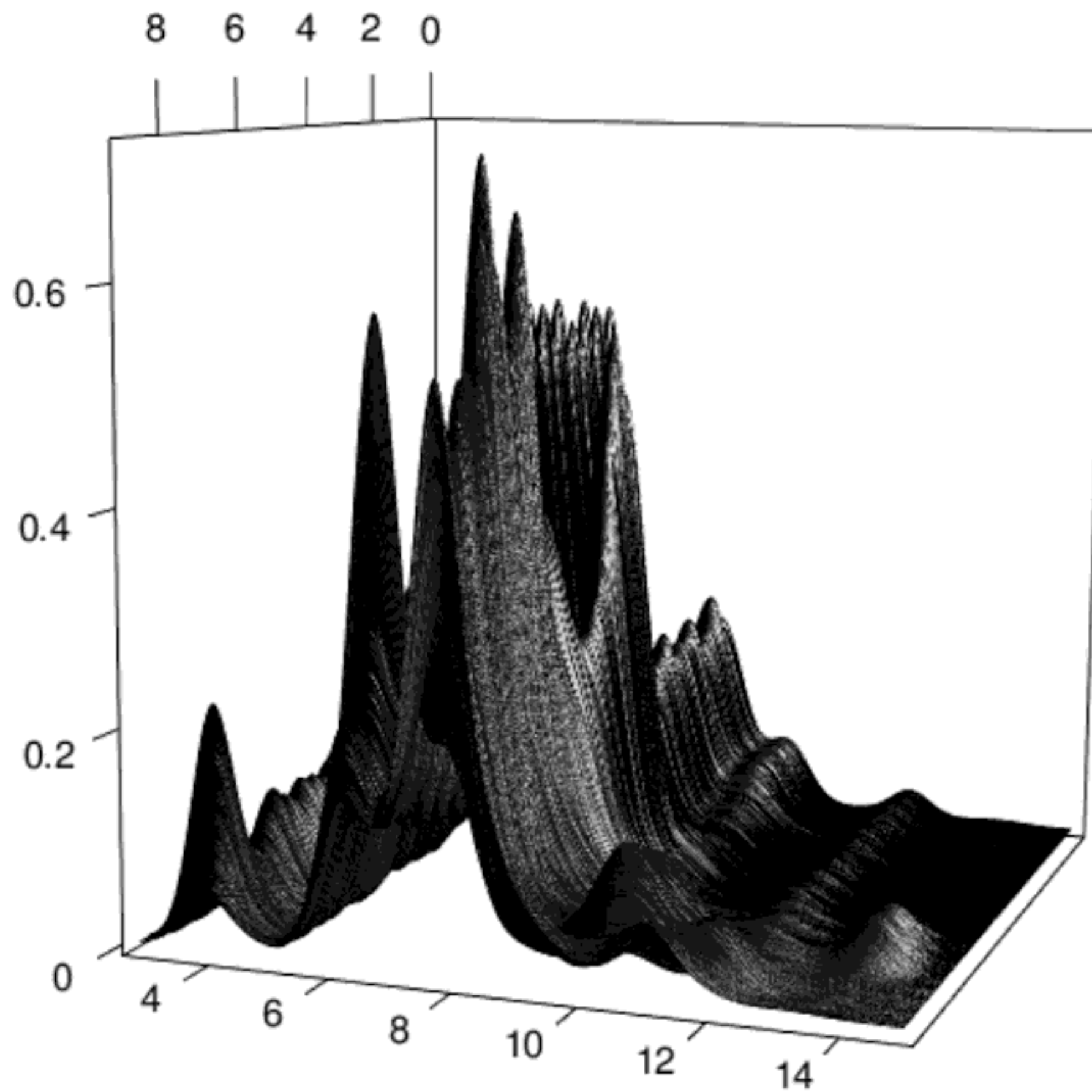


Fig. 6: Streaming kernel density estimate for web traffic from the flows data. The  $x$ -axis (front) correspond to the log of the number of bytes in the flow, the  $y$ -axis (into the page) is time, and the  $z$ -axis is density.

unlabelled, and the task is to group the data into “appropriate” groups; and outlier or abnormality detection, where the data are “normal”, and the computer is tasked to recognise when new data do not fit well with the training data.

There are a myriad of classification algorithms. Recently (Fernández-Delgado *et al.*, 2014), it was observed that from a practical standpoint, there are only a few classifiers that are robust enough to be both easily applicable to a wide range of data sets and at the top of the performance curve. One of the best, if not the best, is random forests (Breiman, 2001a).

The idea of random forests comes from decision trees (Breiman *et al.*, 1984). Think of a physician diagnosing a patient. She may go through a set of rules like: Does the patient exhibit a fever? If not, does the patient complain of abdominal pain? One can imagine building a tree of simple univariate tests of the data, where at each node in the tree, the data are split on the value of the variable, until one reaches a leaf node, where the decision is made based on the class labels of the training data in that node.

This works quite well in practice, and there are a number of software systems that implement variations on this idea. The advantage is that the classifier is easy to understand as a series of if-tests, and a physician can look at the individual decisions to determine whether she believes that the algorithm is computing something that makes sense. The disadvantage is that it is extremely sensitive to the data — changing the training data a little bit, can dramatically change the tree. Also, although the approach works quite well, a single tree is limited in practice and is generally not the best classifier that can be obtained.

The random forest takes advantage of the variability limitation mentioned above. It operates by taking a sample of the data, building a tree on that sample, then repeating many times. The resulting collection of trees (the forest) is then used in a voting scheme to classify new data. In practice, this works amazingly well, as indicated in the reference (Fernández-Delgado *et al.*, 2014). When we apply the random forest to the malware data, we obtain an error of 0.040, far better than the kernel estimator approach. Recall that the data used is simply byte-counts, which we have already noted is a particularly naive set of features, and without a doubt could be improved upon with a bit of domain knowledge, and yet we obtain better than 95% correct classification on the malware task. Note that unlike the nearest neighbour classifier, the random forest does slightly worse on the kernel estimator (an error of 0.064) — it may be that the correlation introduced by the smoothing is harming the performance.

### 3. Manifold Learning

It is generally believed that most real high-dimensional data fall on or near a lower-dimensional structure. Here, “near” is generally considered in a probabilistic sense; crudely, we think of the data as being drawn from a distribution on this structure, with additive (high-dimensional) noise. This structure is the “manifold” of the section title, but technically it need not be a manifold in the strict topological or geometric sense. Manifold learning, also known as manifold discovery, is a collection of techniques for “discovering” this structure, usually by way of an embedding into a lower-dimensional (Euclidean) space.

This idea that there is a lower-dimensional representation of the data that contains the information relevant for inference is the basis of principal component analysis (Jolliffe, 2002), multi-dimensional scaling (Cox and Cox, 2000), feature selection (Guyon and Elisseeff, 2003), projection pursuit (Friedman and Stuetzle, 1981), etc.

Many manifold learning techniques utilise graphs defined on the data (Belkin and Niyogi, 2003; Cayton, 2005; Huo *et al.*, 2007; Pless and Souvenir, 2009), and so we will start with a discussion of graph theory.

### 3.1. Graph theory

A graph is a set  $V$  of vertices (or nodes) and a set of edges  $E \subset V \times V$ . We assume that there are no self loops (if  $(u, v) \in E$  then  $u \neq v$ ). We will also assume the graph is undirected, and so  $(u, v)$  and  $(v, u)$  correspond to the same edge, so the elements of  $E$  are unordered pairs (in spite of the notation). The *order* of a graph  $g = (V, E)$  is  $|V|$ , the number of vertices, and the *size* is  $|E|$ , the number of edges. The adjacency matrix of a graph is the binary matrix  $A = (a_{ij})$  where  $a_{ij} = 1$  if and only if there is an edge from vertex  $i$  to vertex  $j$ . The degree of a vertex is the number of edges incident to it.

One can construct a graph from data by treating the points as vertices and defining edges in terms of proximity. Usually, one defines a distance or dissimilarity measure and uses this to join “close” vertices. The most common such graphs are the  $k$ -nearest neighbour graph — in which each vertex is joined to the  $k$  points nearest to it — and the  $\epsilon$ -ball graph — in which each vertex is joined to all points whose distance is within  $\epsilon$ .

Generally, especially with high-dimensional data, the  $\epsilon$ -ball graph is disconnected. One way to mitigate this without adding an excessive number of edges, is to add in the edges from a  $k$ -nearest neighbour graph, with  $k$  chosen to be minimal such that the resulting graph is connected.<sup>o</sup> Performing this operation with the malware training data results in a graph with 13005 edges. Laying the graph out using a spring embedding algorithm (Kamada and Kawai, 1989) is depicted in Figure 7. Here,  $k = 9$  connects the graph

---

<sup>o</sup>A (possibly) better approach is to compute the minimal spanning tree first, then add the edges from the  $\epsilon$ -ball graph.

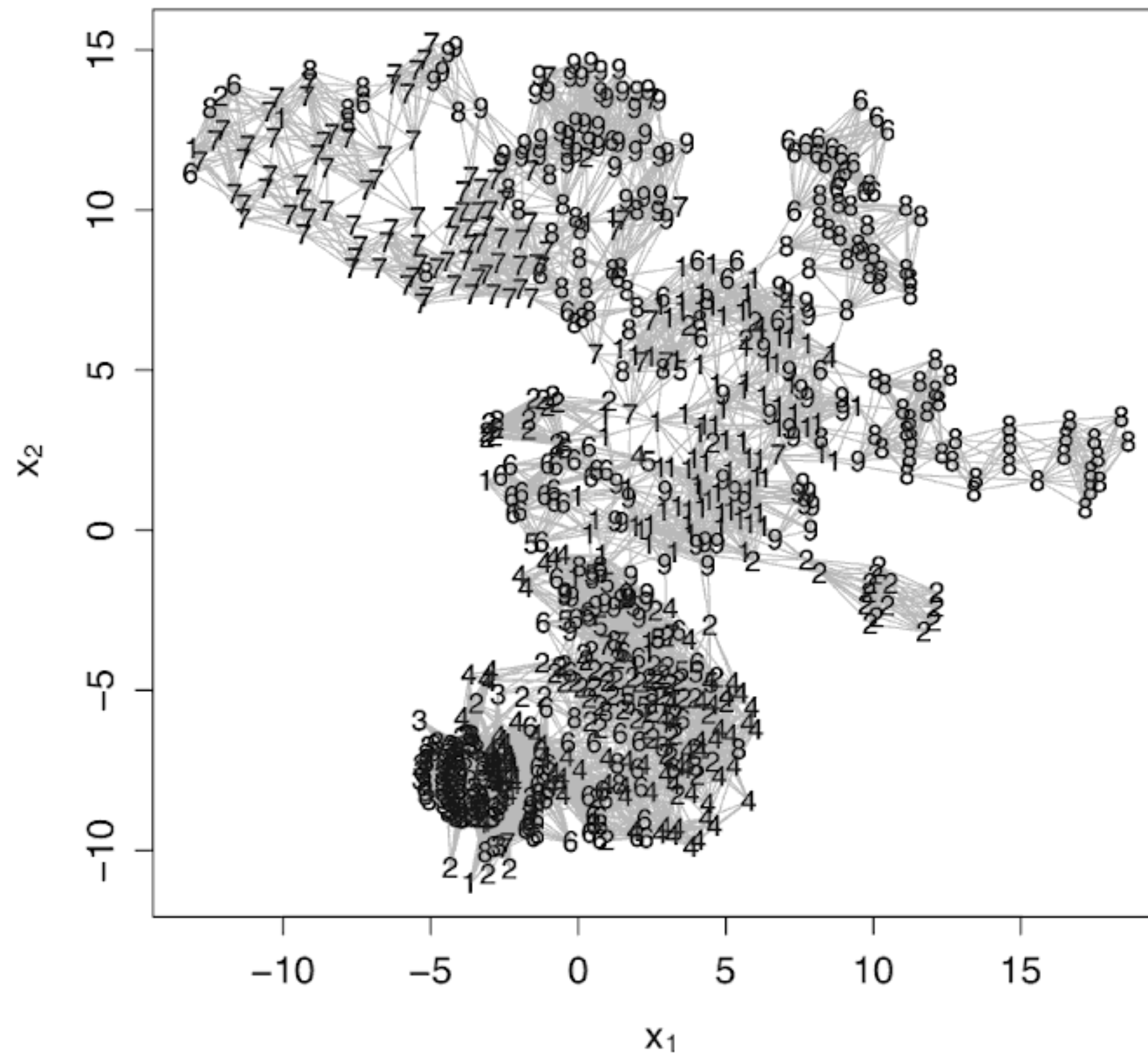


Fig. 7: Two-dimensional layout of the connected  $\epsilon = .01$  ball graph on the malware training data. Numbers correspond to the class of the observation.

(the  $k = 9$  nearest neighbour graph has two connected components). The 9-nearest neighbour graph is depicted in Figure 8.

### 3.2. Dimensionality reduction

The graph embedding discussed above is one way of reducing the dimension of the data. However, using graph layout algorithms is clearly not what is needed. These algorithms are designed for two- and three-dimensional layouts, and are meant to be visually appealing, or have other measures of quality, that are not necessarily well suited to the inference task at hand.

Instead, we wish to use the graph to provide an embedding into  $\mathbb{R}^d$  for any (reasonable)  $d$ , and further, we'd like this embedding to be well suited for inference. The Isomap (Tenenbaum *et al.*, 2000) algorithm is one method worth considering.

Given a graph,  $g$ , we compute the shortest path distance between each pair of vertices. This produces an inter point distance matrix between each pair of our observations (treated as vertices of the graph). One then applies multi-dimensional scaling (Borg and Groenen, 1997) to this matrix to produce the embedding.

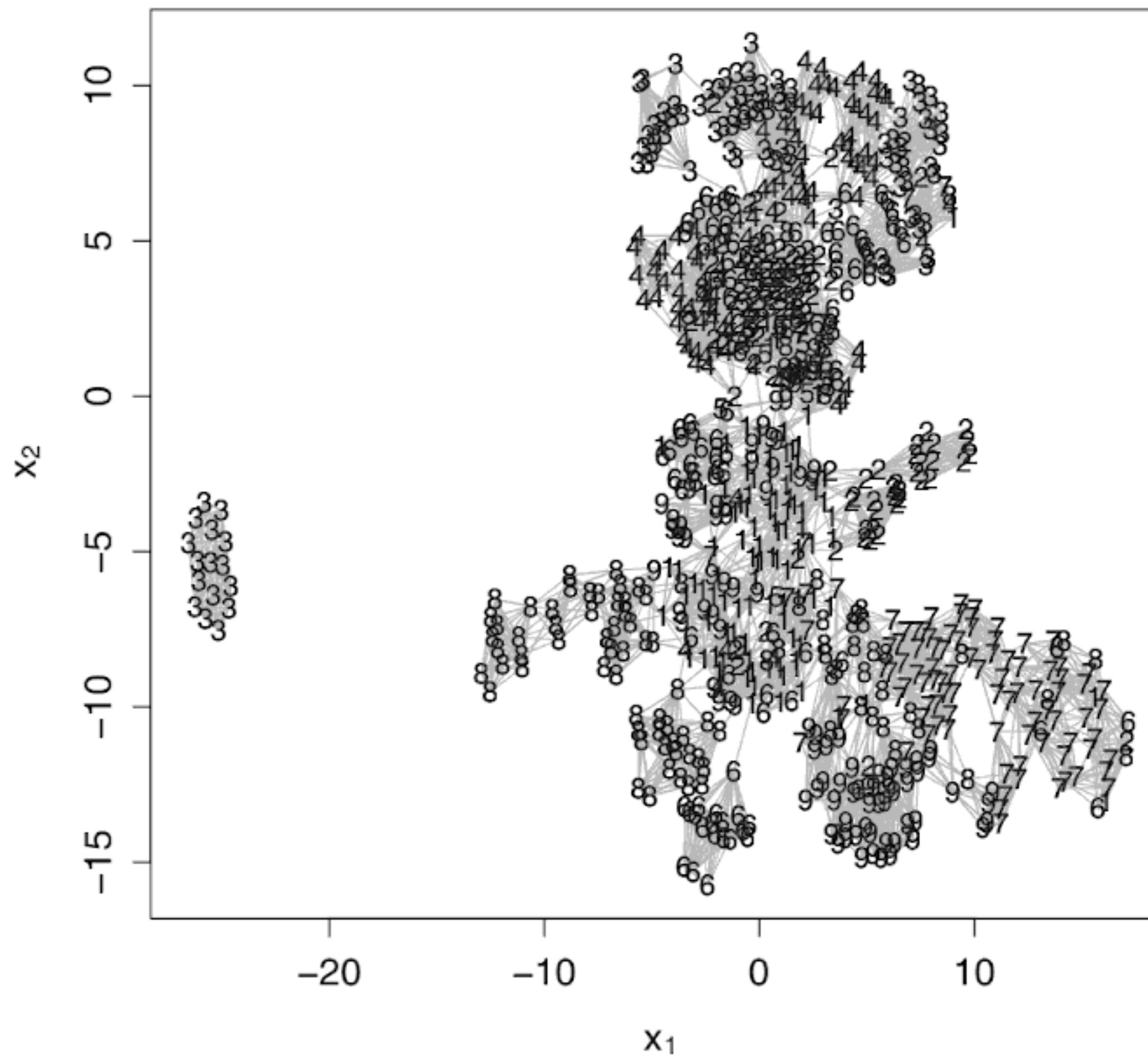


Fig. 8: Two-dimensional layout of the  $k = 9$  nearest neighbour graph on the malware training data. Numbers correspond to the class of the observation.

Figure 9 shows the Isomap embedding using the connected  $\epsilon$ -ball graph. The 9-nearest neighbour graph is problematic, since it is not connected. There are two approaches: embed each component separately, or increase  $k$  until the graph is connected. Since the 11-nearest neighbour graph is connected, we use this in Figure 10.

Another popular method of embedding is the Laplacian eigenmap. The Laplacian of a graph is  $L = D - A$  where  $A$  is the adjacency matrix and  $D$  is the diagonal matrix of the degrees of the vertices (the row-sums of  $A$ ). It is well known that  $L$  is positive semi-definite with a 0 eigenvalue for each connected component of the graph. Assuming the graph is connected, the  $d$  eigenvectors associated with the  $d$  smallest non-zero eigenvalues is often used as an embedding of the graph into  $\mathbb{R}^d$ . The normalised Laplacian  $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$  is also often used, and empirically seems to provide better embeddings for most classification and clustering tasks.

An interesting model for graphs is the latent position model (Hoff *et al.*, 2002), in which the edges of a graph depend probabilistically on their relationship in a latent (unobserved) “social space”. A simple version of this is the random dot product graph (RDPG) (Marchette and Priebe, 2008; Nickel, 2008; Scheinerman and Tucker, 2010). The model is as follows. Assume

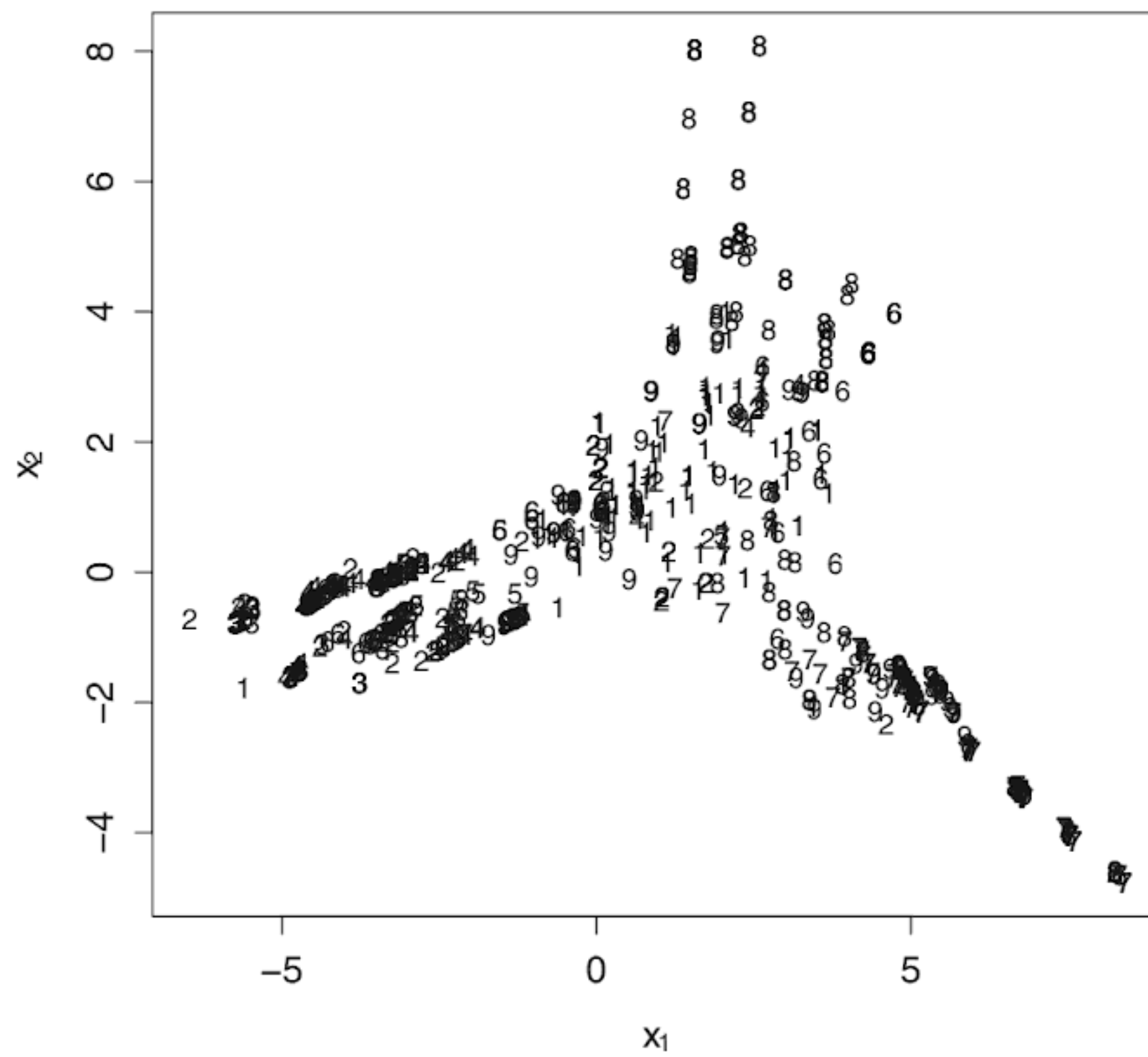


Fig. 9: Two-dimensional Isomap of the connected  $\epsilon = .01$  ball graph on the malware training data. Numbers correspond to the class of the observation.

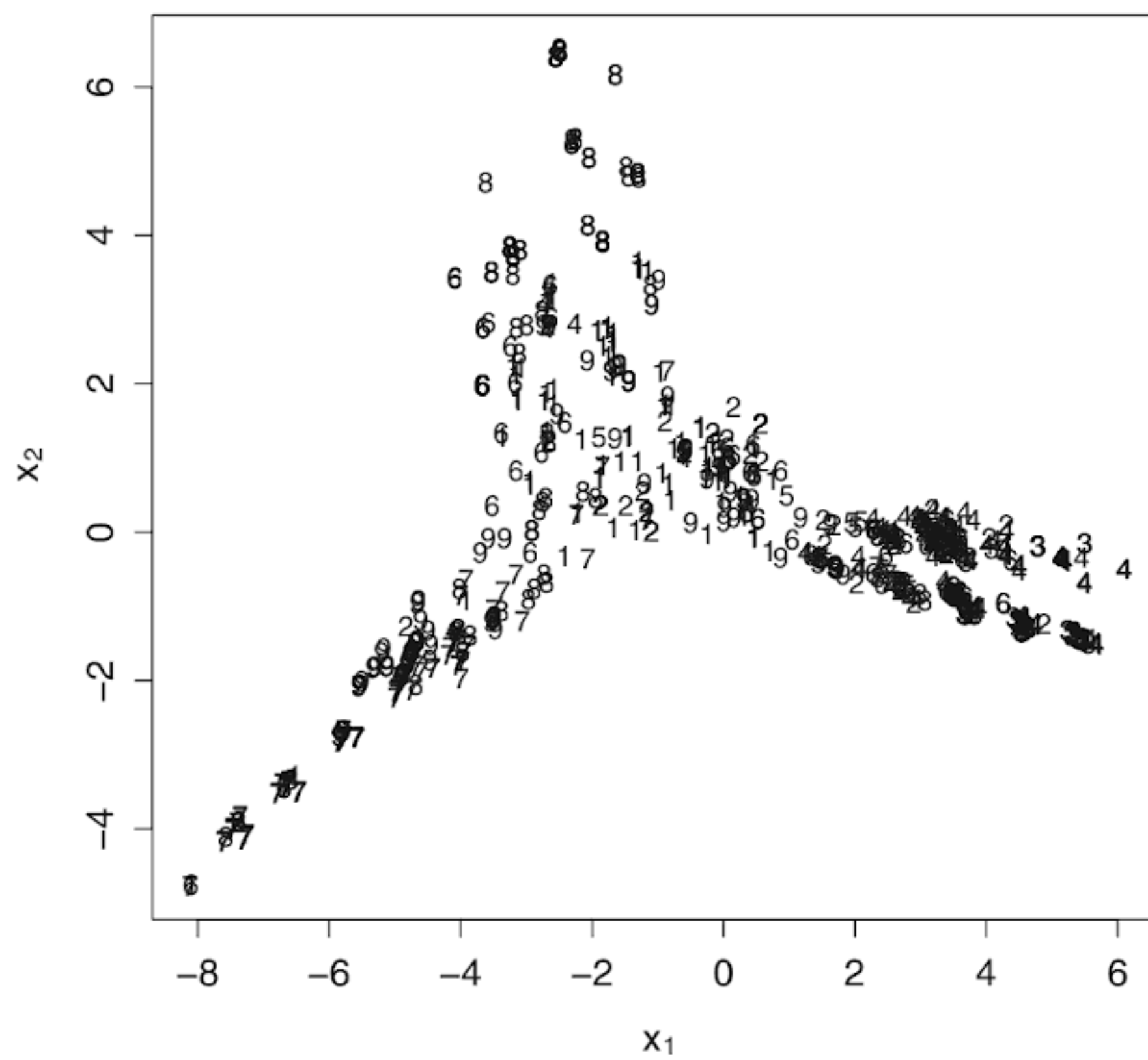


Fig. 10: Two-dimensional Isomap embedding of the  $k = 11$  nearest neighbour graph on the malware training data. Numbers correspond to the class of the observation.

there are (unobservable) vectors  $(v_1, \dots, v_n) \in \mathbb{R}^d$ , one per vertex, with the property that all pairwise dot products between vectors lie in  $[0, 1]$ . The graph is generated according to

$$P((i, j) \in E) = v_i^t v_j.$$

The random dot product graph has a couple of nice properties. In particular, it can utilise the spectral theorem to estimate the vectors. Recall that the spectral theorem for symmetric square matrices  $M$  can be written as:

$$M = U \Lambda U^t = (U \Lambda^{-\frac{1}{2}})(U \Lambda^{-\frac{1}{2}})^t. \quad (5)$$

The columns of  $U$  are the eigenvectors, and  $\Lambda$  is the diagonal matrix of eigenvalues, and  $U$  and  $\Lambda$  are real valued. We'll assume the eigenvalues are in decreasing size, so the first column of  $U$  corresponds to the largest eigenvalue. Further, the best (in Frobenius norm) representation of  $M$  with  $(n \times d)$ -dimensional matrices is to take the first  $d$  eigenvectors in (5).

Given a graph, the obvious thing to do is to use (5) applied to the adjacency matrix, taking the top  $d$  eigenvectors/values. There are two problems with this. First is the fact that the adjacency matrix is hollow — the diagonal is structurally zero — and so the statement about the “best  $d$ -dimensional representation” isn't quite right. Second is that there's no reason to think the adjacency matrix is positive, and so there is a very real possibility that the square root is going to result in imaginary numbers if  $d$  is too large.

The second observation can be eliminated by noticing that the eigenvalues are simply a coordinate-wise scaling, and so one could simply use the eigenvectors, since the purpose is to produce an embedding of the data, not necessarily a good estimate of the latent vectors.<sup>P</sup> The first problem is a bit trickier. If we really want an estimate of the latent vectors, we don't want the diagonal of the resultant dot-product matrix to be 0. What we'd like to do is “fill in” the diagonal matrix with “the right” values — the lengths of the latent vectors. Adding a diagonal matrix to the adjacency matrix changes only the eigenvalues, not the eigenvectors, and so we need an estimate of the lengths of the latent vectors. A reasonable

---

<sup>P</sup>Alternatively, we could simply choose  $d$  to be small enough that the first  $d$  eigenvalues are positive.

estimate is  $D/(n - 1)$  where  $n$  is the order of the graph. So, we operate on  $A + D/(n - 1)$ .

If the graph is a stochastic block model (Athreya *et al.*, 2016), and one chooses  $k$  to be the number of blocks (actually, the rank of the probability matrix), the resultant embedded points are distributed as a mixture of Gaussians.<sup>9</sup> Note that this statement is not changed by whether the vectors are or are not scaled by the eigenvalues — since this is only a scaling of the embedding, the mixture model is still the appropriate model.

Figure 11 depicts the RDPG embedding. As with the other embeddings, this is misleading, since we are depicting a two-dimensional representation, and it is unlikely that this is the intrinsic dimension of the data. However, it illustrates the idea of the RDPG embedding.

### 3.3. Fusion

Often, one has more than one way of looking at an object — more than one sensor or more than one way to construct features. For example, with the

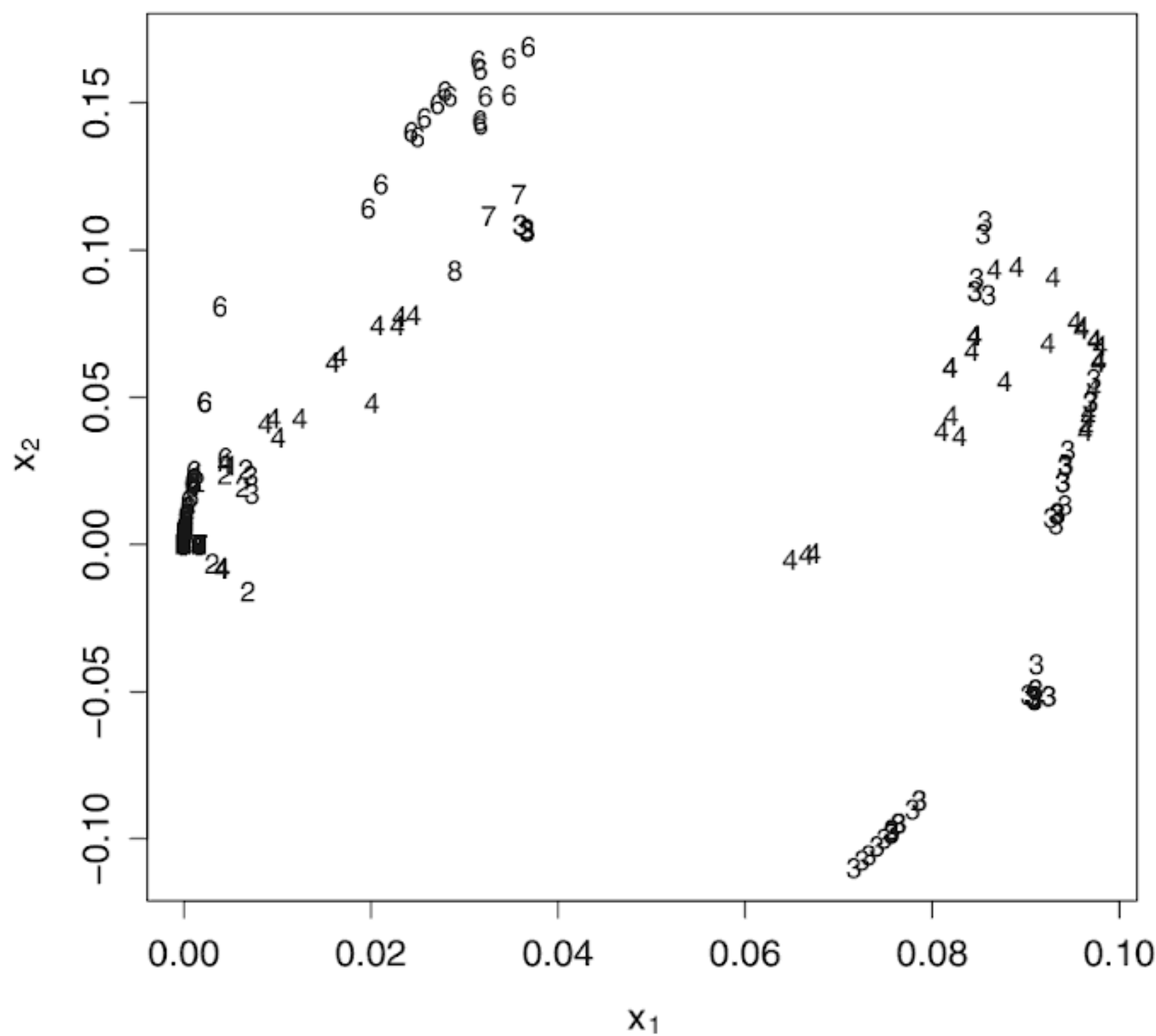


Fig. 11: Random dot product embedding of the connected  $\epsilon$ -ball graph on the malware training data, with  $\epsilon = 0.1$ . Numbers correspond to the class of the observation.

<sup>9</sup>This is an asymptotic result, and there are a couple of technical details, but from a practical standpoint, a mixture of normals will be a good model for the embedded data.



malware data, one can look at the hex dump of the bytes (as we have done here) or one could look at a decompilation of the programme. Each view of the data can produce features, and we would like to “fuse” these features into a single representation of the data.

The obvious way to do this is to simply stick the features together. That is, if we compute the byte-count histogram from the hex dump and the word-count histogram from the decompilation, we can append these columns and use this (massively large) number of features as our feature set. This is, generally speaking, a bad idea. The curse of dimensionality (Jain *et al.*, 2000; Scott, 1992) says that the noise added by these new features can overcome any extra information inherent in the features. See Trunk (1979) for an excellent illustration of this fact.

One method for “fusing” information can be done at the inter point distance matrix level. Compute the distances from data from the first sensor (the byte-count histogram), producing  $D_1$  and from the second sensor (the word-count histogram), producing  $D_2$ . Combine these distances into an *omnibus* inter point distance matrix  $D$  and use multi-dimensional scaling to embed these into a manageable space for inference. There are several ways to do this, one of which (Ma *et al.*, 2012; Priebe *et al.*, 2013) is to form the matrix:

$$D = \begin{pmatrix} D_1 & W_\lambda \\ W_\lambda & D_2 \end{pmatrix},$$

where  $W_\lambda = \lambda D_1 + (1 - \lambda) D_2$ .

Note that the above leads to a similar approach if the distance matrices are replaced by the adjacency matrices of graphs defined on the data from the two sensors. This leads to a “joint” manifold discovery approach and is the subject of further research.

#### 4. Topological Data Analysis

The field of algebraic topology seeks to understand topological spaces — such as the support regions of distributions — through the study of invariants computed on the spaces. These measures are invariant to smooth distortions of the space.

Topological data analysis (TDA) utilises the tools of algebraic topology for the analysis of data by first defining a sequence of topological spaces associated with the data and computing certain invariants on these spaces to learn information about the global structure of the data. Information about TDA can be found in Carlsson (2009), and in the book by Ghrist (2014).

Usually one uses TDA to explore a data set of points in some space, and extract features that describe some of the characteristics of the points, or more generally, the distribution from which the points were drawn. I will instead look at applying TDA to the malware data, where each piece of malware is viewed as a set of points, and TDA is used to extract features of the malware.

Much of the topological background we will use can be found in Greenberg and Harper (1981), as well as many other standard algebraic topology books. We will define the basic tools that will be used for TDA, then discuss how these may be used to make inferences about the classification boundary.

#### 4.1. *Simplicial homology*

A (geometric) simplex of dimension  $d$  is a set of  $d + 1$  points in relative position, see Figure 12. So, a 0-simplex is a point, a 1-simplex a line segment, a 2-simplex a triangle, and so on. We will write a  $k$ -simplex as  $k + 1$  vertices:  $[v_0, \dots, v_k]$ , and any subset of  $k$  vertices is called a  $(k - 1)$ -face. We will say that a  $k$ -simplex is  $k$ -dimensional.

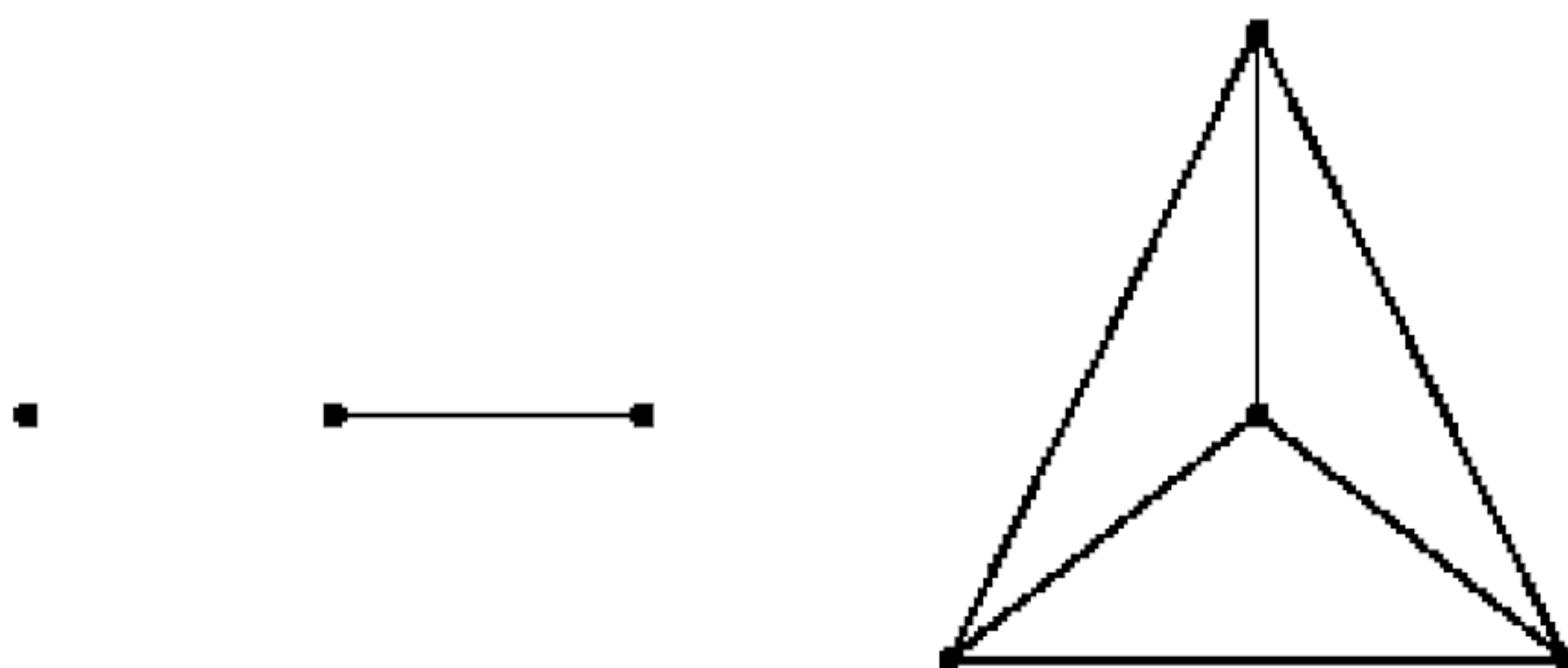


Fig. 12: Examples of a 0-simplex, a 1-simplex, and a 3-simplex (the figure on the right should be considered as a solid three-dimensional tetrahedron).

A simplicial complex is a collection  $\mathcal{S}$  of simplices that satisfies the following conditions:

- (1) If  $\sigma \in \mathcal{S}$ , then so are the faces of  $\sigma$ .
- (2) If  $\sigma_1, \sigma_2 \in \mathcal{S}$  are  $k$  simplices, then either they are disjoint or they intersect in a lower-dimensional simplex which is a face of both.

We write the set of  $k$ -simplices of  $\mathcal{S}$  as  $S_k$ . We impose an orientation on the simplices by giving a positive orientation to vertices in lexicographic order, and asserting that any odd permutation of a  $k$ -simplex  $[v_0, \dots, v_k]$  reverses the orientation. Thus,  $[v_0, \dots, v_i, \dots, v_j, \dots, v_k] = -[v_0, \dots, v_j, \dots, v_i, \dots, v_k]$ .

All of this can be abstracted by defining an abstract simplicial complex as a collection  $\Delta$  of non-empty finite sets such that  $Y \subset X \in \Delta$  implies  $Y \in \Delta$ . In this case,  $Y$  is called a face of  $X$ .

Given a simplicial complex, we form the chain complex as follows. For each  $S_k$  of  $k$ -simplices, we form the vector space (over some field, usually finite)  $C_k$  whose basis is the set of  $k$ -simplices in  $S_k$ . This can be thought of as all formal sums of the form  $\sum a_i \sigma_i$  for  $a_i \in \mathbb{Z}_p$ ,  $\sigma_i \in S_k$ . The boundary of a  $k$  simplex is the set of  $k - 1$  simplices formed by deleting a vertex. This gives rise to a boundary map  $\partial_k : C_k \rightarrow C_{k-1}$  for  $k \geq 1$  as

$$\partial[v_0, \dots, v_k] = \sum_i (-1)^i [v_0, \dots, v_{i-1}, v_{i+1}, \dots, v_k].$$

The chain complex is the set  $\{C_k, \partial_k\}$ , often presented as

$$\cdots \xrightarrow{\partial_{k+1}} C_k \xrightarrow{\partial_k} C_{k-1} \cdots \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0.$$

It is easy to check that  $\partial_k \partial_{k+1} = 0$ . We call the kernel of  $\partial_k$  the  $k$ -cycles, written  $Z_k$ , and the image of  $\partial_{k+1}$  the  $k$ -boundaries, written  $B_k$ . The homology is the quotient  $H_k = Z_k / B_k$ .

The rank of the  $k$ th homology is referred to as the  $k$ th Betti number, and it is these numbers that we will compute. Given a space, one constructs a simplicial complex, for example by triangulating the space, and then computes the homology. This is a topological invariant, meaning that continuous distortions of the space do not change the topology, and in particular do not change the Betti numbers.

In some sense, the homology of a space tells something about the  $k$ -dimensional holes, or voids, in the space. For example,  $\text{Betti}_0$  is the number of connected components of the space. For an  $n$ -sphere, the Betti numbers are 1 for  $k = 0, n$  and zero otherwise. The homology of the 2-torus  $S^1 \times S^1$  is 2 for  $k = 1$  (note that this corresponds to the two circles which generate the torus) and 1 for  $k = 0, 2$ , zero elsewhere.

Homology has an interesting connection to the Euler characteristic. One defines the Euler characteristic as

$$\chi(X) = \sum_{j=0}^n (-1)^j \text{Betti}_j(X). \quad (6)$$

This is equivalent to the standard Euler characteristic one learns in grade school, extended to general topological spaces and higher dimensions.

## 4.2. Homology of data

The above is all in the context of an abstract space — these abstract simplicies and chain complexes. How does one take a data set and apply any of this to learn anything about the data?

Just as with manifold discovery, the idea is to construct a graph on the data, such as a  $k$ -nearest neighbour graph, an  $\epsilon$ -ball graph, or some other graph that encodes the local structure of the data. This graph then defines a simplicial complex through one of a number of mechanisms. The simplest is to treat each clique on  $k + 1$  vertices as a  $k$ -simplex.

Given a data set, we construct a graph from which we construct a simplicial complex and compute the homology of the complex, which then corresponds to the homology of the data. This procedure is highly dependent on the choices made to define the graph. There are two obvious approaches to consider. The first is to construct a graph that is defined on data without regard to any parameters such as  $k$  or  $\epsilon$ . Relative neighbourhood graphs or Gabriel graphs (Marchette, 2004) are two such graphs. The second is to consider a sequence of graphs (such as several values of  $k$  or  $\epsilon$ ) and look at how the homology changes as we change these values. This latter approach leads to the idea of persistent homology: those Betti numbers that “persist” across scales are ones that are indicative of true structure, while those that “come and go” are driven by noise and can be ignored.

### 4.3. Euler trajectory

We define the Euler Trajectory as the Euler characteristic as a function of  $\epsilon$  in the  $\epsilon$ -ball graph of the persistent homology. Figure 13 shows the Euler trajectories for the nine families of malware. These trajectories are computed from the persistent homology of the two-dimensional kernel estimators, using the TDA (Fasy *et al.*, 2015) package in R (Ihaka and Gentleman, 1996). There is some difference in the trajectories between the families, but for these data the difference is not large.

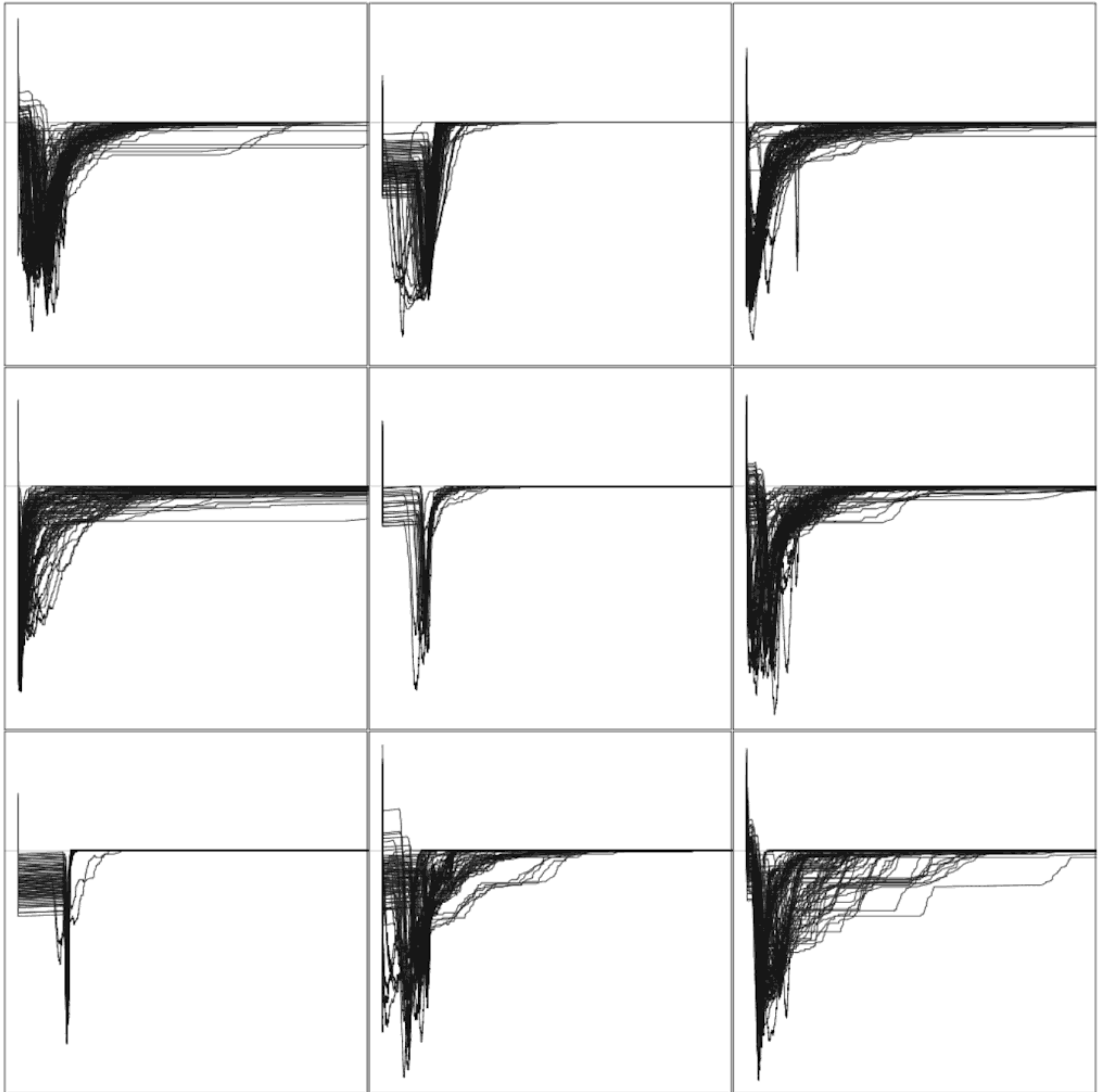


Fig. 13: The Euler trajectories for each of the nine classes of malware. In this figure  $\epsilon \in [0, 0.0001]$ .

There are several distances between persistent homologies in the literature. The bottleneck distance (Cohen-Steiner *et al.*, 2007; Veltkamp, 2001) looks at all bijections between the pairs of points in the diagram and computes the minimum of the maximum distances between points and their images. Applying multi-dimensional scaling to this distance matrix into  $\mathbb{R}^5$  results in Figure 14. This gives another view of the data that may provide insight into its structure, and can provide a set of features to use for subsequent inference.

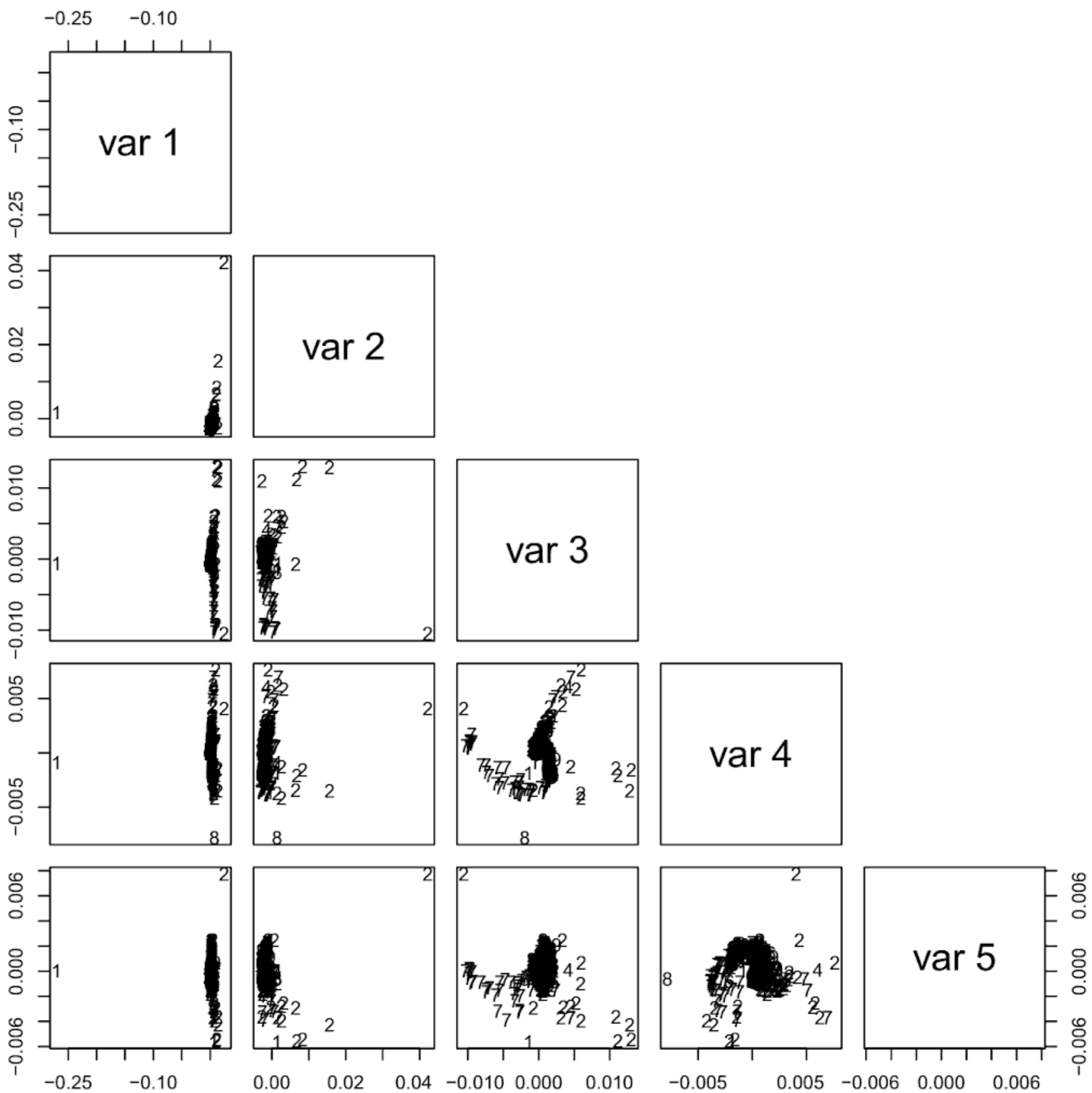


Fig. 14: The multi-dimensional scaling projection of the distances between the persistent homologies for the malware data. Plotting character indicates malware family.

## 5. Discussion

This has been a very high-level and idiosyncratic look at some techniques from computational statistics and statistical pattern recognition. There is a huge literature in feature selection, manifold discovery, and machine learning that we have only very briefly touched on. It is hoped that this will provide some suggestions for directions to look for future work, and some references for methods that may be of value for the analysis of computer security data.

One of the take-aways from this work is that very simple ideas can be very powerful. The simplistic features extracted from the malware data turn out to go a long way towards providing the information necessary for classification. The very simple idea of averaging — computing the sample mean — is used in the kernel estimator, the streaming data algorithms and the random forest.

Hand (2006) notes the power of simple classifiers and warns against putting too much effort into squeezing out the last bit of performance on a given data set. He points out many problems such as: changes in the underlying distribution of the data; training/testing data that is not representative; model misspecification; uncertainty in the classification labels and issues of interpretability.

With this said, it is important to remember that some things, cyber-security being one of them, are inherently complicated, and we should be wary of oversimplifying. Careful thought needs to be put into incorporating domain knowledge into the models, and the models need to be adaptable to the changing environment — here streaming methods can be helpful. Interpretation and validation of the model, particularly in the case of random forests, can be quite difficult, and we need more tools to address these issues.

The field of deep learning (neural networks) has been completely missed in this discussion. This is not to denigrate it or to suggest that it is not important. There is reason to believe that the techniques that are currently available for training these deep networks may lead to very useful algorithms and solutions for the problems of cyber-security. These methods do suffer from some of the same issues of interpretability and validation that random forests have, so they should not be viewed as a panacea.

## Acknowledgements

This work was funded in part by the Naval Surface Warfare Center (NSWC) In-House Laboratory Independent Research (ILIR) programme.

## References

- Aggarwal, C. C. and Zhai, C. (2012). *Mining Text Data*, Springer Science & Business Media, LLC, New York, USA.
- Agresti, A. (1990). *Categorical Data Analysis*, Wiley, New York.
- Athreya, A., Priebe, C. E., Tang, M., Lyzinski, V., Marchette, D. J. and Sussman, D. L. (2016). A limit theorem for scaled eigenvectors of random dot product graphs, *Sankhya A* **78**, 1, pp. 1–18.
- Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation, *Neural Computation* **15**, 6, pp. 1373–1396.
- Blei, D. M. and Lafferty, J. D. (2009). *Topic Models*, Chapman & Hall/CRC.
- Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003). Latent Dirichlet allocation, *Journal of Machine Learning Research* **3**, pp. 993–1022.
- Borg, I. and Groenen, P. (1997). *Modern Multidimensional Scaling*, Springer, New York.
- Breiman, L. (2001a). Random forests, *Machine Learning* **45**, 1, pp. 5–32.
- Breiman, L. (2001b). Statistical modeling: The two cultures, *Statistical Science* **16**, 3, pp. 199–231.
- Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and Regression Trees*, Wadsworth & Brooks, Monterey, CA.
- Carlsson, G. (2009). Topology and data, *Bulletin of the American Mathematical Society* **46**, 2, pp. 255–308.
- Caudle, K. A., Karlsson, C. and Pyeatt, L. D. (2015). Using density estimation to detect computer intrusions, in *Proceedings of the 2015 ACM International Workshop on Security and Privacy Analytics*, pp. 43–48.
- Cayton, L. (2005). Algorithms for manifold learning, *University of California at San Diego Technical Report*, pp. 1–17.
- Cohen-Steiner, D., Edelsbrunner, H. and Harer, J. (2007). Stability of persistence diagrams, *Discrete & Computational Geometry* **37**, 1, pp. 103–120.
- Cox, T. F. and Cox, M. A. (2000). *Multidimensional Scaling*, CRC Press, Boca Raton, FL.
- Donoho, D. (2015). 50 years of data science, in *Tukey Centennial Workshop*, Princeton NJ.
- Fasy, B. T., Kim, J., Lecci, F., Maria, C., included GUDHI is authored by C. Maria, V. R. T., by D. Morozov, D. and PHAT by U. Bauer, M. K. J. R. (2015). *TDA: Statistical Tools for Topological Data Analysis*, URL: <https://CRAN.R-project.org/package=TDA>, R package, Version 1.4.1.
- Fernández-Delgado, M., Cernadas, E., Barro, S. and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research* **15**, pp. 3133–3181.
- Fraley, C. and Raftery, A. E. (2002). Model-based clustering, discriminant analysis and density estimation, *Journal of the American Statistical Association* **97**, pp. 611–631.



- Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression, *Journal of the American Statistical Association* **76**, 376, pp. 817–823.
- Ghrist, R. (2014). *Elementary Applied Topology*, Createspace Independent Publishing Platform.
- Greenberg, M. J. and Harper, J. R. (1981). *Algebraic Topology: A First Course*, CRC Press, Boca Raton, FL.
- Gupta, V. and Lehal, G. S. (2009). A survey of text mining techniques and applications, *Journal of Emerging Technologies in Web Intelligence* **1**, 1, pp. 60–76.
- Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection, *Journal of Machine Learning Research* **3**, pp. 1157–1182.
- Hand, D. J. (2006). Classifier technology and the illusion of progress, *Statistical Science* **21**, 1, pp. 1–14.
- Hoff, P. D., Raftery, A. E. and Handcock, M. S. (2002). Latent space approaches to social network analysis, *Journal of the American Statistical Association* **97**, pp. 1090–1098.
- Huo, X., Ni, X. S. and Smith, A. K. (2007). A survey of manifold-based learning methods, in *Recent Advances In Data Mining Of Enterprise Data*, pp. 691–745, World Scientific, Singapore.
- Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics, *Journal of Computational and Graphical Statistics* **5**, 3, pp. 299–314.
- Jain, A. K., Duin, R. P. W. and Mao, J. (2000). Statistical pattern recognition: A review, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**, 1, pp. 4–37.
- Jolliffe, I. T. ed. (2002). *Principal Component Analysis*, Springer Verlag, New York.
- Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs, *Information Processing Letters* **31**, 1, pp. 7–15.
- Kent, A. D. (2016). Cyber security data sources for dynamic network research, in N. Adams and N. Heard. eds., *Dynamic Networks and Cyber-Security*, Vol. 1, p. 37, World Scientific, Singapore.
- Ma, Z., Marchette, D. J. and Priebe, C. E. (2012). Fusion and inference from multiple data sources in a commensurate space, *Statistical Analysis and Data Mining* **5**, 3, pp. 187–193.
- Marchette, D. J. (2004). *Random Graphs for Statistical Pattern Recognition*, John Wiley & Sons, New York.
- Marchette, D. J. and Priebe, C. E. (2008). Predicting unobserved links in incompletely observed networks, *Computational Statistics and Data Analysis* **52**, pp. 1373–1386.
- Marchette, D. J., Priebe, C. E., Rogers, G. W. and Solka, J. L. (1996). Filtered kernel density estimation, *Computational Statistics* **11**, 2, pp. 95–112.
- McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions*, Wiley, New York.
- McLachlan, G. J. and Peel, D. (2000). *Finite Mixture Models*, Wiley, New York.
- Moore, A. (1998). Very fast EM-based mixture model clustering using multiresolution kd-trees, [citeseer.nj.nec.com/moore98very.html](http://citeseer.nj.nec.com/moore98very.html).
- Nickel, C. L. M. (2008). *Random Dot Product Graphs: A Model For Social Networks*, Ph.D. thesis, Johns Hopkins University, Baltimore, MD.
- Pless, R. and Souvenir, R. (2009). A survey of manifold learning for images, *IPSN Transactions on Computer Vision and Applications* **1**, pp. 83–94.

- Priebe, C. E. (1994). Adaptive mixture density estimation, *Journal of the American Statistical Association* **89**, pp. 796–806.
- Priebe, C. E., Marchette, D. J., Ma, Z. and Adali, S. (2013). Manifold matching: Joint optimization of fidelity and commensurability, *Brazilian Journal of Probability and Statistics* **27**, pp. 377–400.
- Scheinerman, E. R. and Tucker, K. (2010). Modeling graphs using dot product representations, *Computational Statistics* **25**, 1, pp. 1–16.
- Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*, Wiley, New York.
- Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, New York.
- Tenenbaum, J. B., de Silva, V. and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction, *Science* **290**, pp. 2319–2323.
- Trunk, G. V. (1979). A problem of dimensionality: A simple example, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1**, 3, pp. 306–307.
- Veltkamp, R. C. (2001). Shape matching: Similarity measures and algorithms, in *SMI 2001 International Conference on Shape Modeling and Applications*, pp. 188–197.
- Wand, M. P. and Jones, M. C. (1994). *Kernel Smoothing*, Chapman and Hall/CRC, London.
- Wegman, E. J. and Marchette, D. J. (2003). On some techniques for streaming data: A case study of internet packet headers, *Journal of Computational and Graphical Statistics* **12**, pp. 893–914.

This page is intentionally left blank