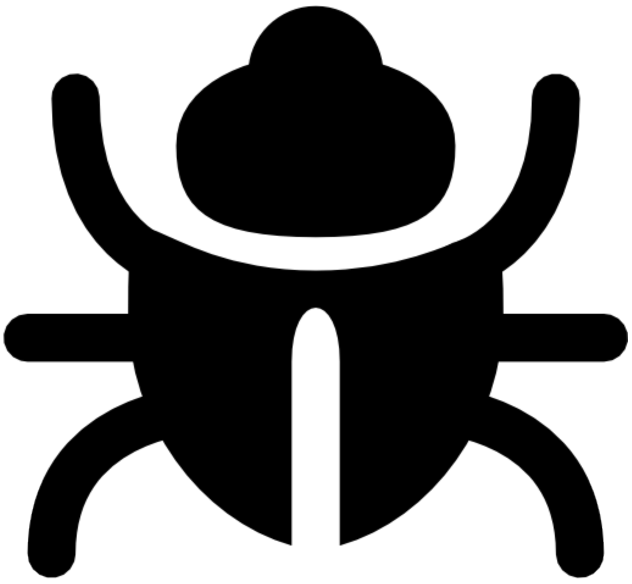
CHALMERS
UNIVERSITY OF TECHNOLOGY



Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning

Master's thesis in Complex Adaptive Systems

MICHEL EDKRANTZ

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Master's thesis 2015

Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning

MICHEL EDKRANTZ

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning
MICHEL EDKRANTZ

Supervisors
Alan Said, PhD, Recorded Future
Christos Dimitrakakis, O. Docent, Department of Computer Science

Examiner
Devdatt Dubhashi, Prof., Department of Computer Science

Master's Thesis 2015
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Cyber Bug. Thanks to Icons8 for the icon `icons8.com/web-app/416/Bug`. Free for commercial use.

Typeset in LaTeX
Printed by Reproservice (Chalmers printing services)
Gothenburg, Sweden 2015

Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning
MICHEL EDKRANTZ
m.edkrantz@gmail.com
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Every day there are some 20 new cyber vulnerabilities released, each exposing some software weakness. For an information security manager it can be a daunting task to keep up and assess which vulnerabilities to prioritize to patch. In this thesis we use historic vulnerability data from the National Vulnerability Database (NVD) and the Exploit Database (EDB) to predict exploit likelihood and time frame for unseen vulnerabilities using common machine learning algorithms. This work shows that the most important features are common words from the vulnerability descriptions, external references, and vendor products. NVD categorical data, Common Vulnerability Scoring System (CVSS) scores, and Common Weakness Enumeration (CWE) numbers are redundant when a large number of common words are used, since this information is often contained within the vulnerability description. Using several different machine learning algorithms, it is possible to get a prediction accuracy of 83% for binary classification. The relative performance of multiple of the algorithms is marginal with respect to metrics such as accuracy, precision, and recall. The best classifier with respect to both performance metrics and execution time is a linear time Support Vector Machine (SVM) algorithm. The exploit time frame prediction shows that using only public or publish dates of vulnerabilities or exploits is not enough for a good classification. We conclude that in order to get better predictions the data quality must be enhanced. This thesis was conducted at Recorded Future AB.

Keywords: Machine Learning, SVM, cyber security, exploits, vulnerability prediction, data mining

Acknowledgements

I would first of all like to thank Staffan Truvé, for giving me the opportunity to do my thesis at Recorded Future. Secondly I would like to thank my supervisor Alan Said, at Recorded Future, for supporting me with hands on experience and guidance in machine learning. Moreover, I would like to thank my other co-workers at Recorded Future, especially Daniel Langkilde and Erik Hansbo from the Linguistics team.

At Chalmers I would like to thank my academic supervisor Christos Dimitrakakis, who introduced me to some of the algorithms, workflows, and methods. I would also like to thank my examiner Prof. Devdatt Dubhashi, whose lectures in Machine Learning provided me with a good foundation in the algorithms used in this thesis.

Michel Edkrantz
Göteborg, May 4th 2015

Contents

Abbreviations

Cyber-related

CAPEC	Common Attack Patterns Enumeration and Classification
CERT	Computer Emergency Response Team
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
EDB	Exploits Database
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
RF	Recorded Future
WINE	Worldwide Intelligence Network Environment

Machine learning-related

FN	False Negative
FP	False Positive
kNN	k-Nearest-Neighbors
ML	Machine Learning
NB	Naive Bayes
NLP	Natural Language Processing
PCA	Principal Component Analysis
PR	Precision Recall
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
SVC	Support Vector Classifier
SVD	Singular Value Decomposition
SVM	Support Vector Machines
SVR	Support Vector Regressor
TN	True Negative
TP	True Positive

1 Introduction

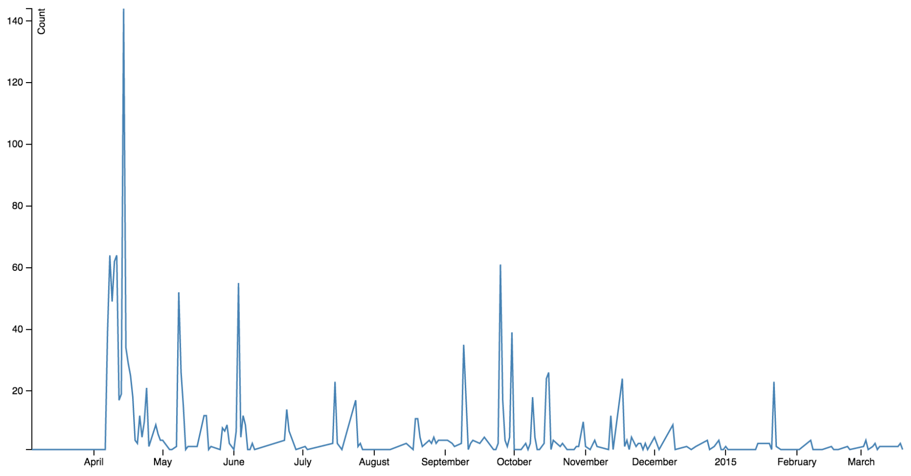


Figure 1.1: Recorded Future Cyber Exploits events for the cyber vulnerability Heartbleed.

Every day there are some 20 new cyber vulnerabilities released and reported on open media, e.g. Twitter, blogs, and news feeds. For an information security manager it can be a daunting task to keep up and assess which vulnerabilities to prioritize to patch (Gordon, 2015).

Recoded Future[1] is a company focusing on automatically collecting and organizing news from 650,000 open Web sources to identify actors, new vulnerabilities and emerging threat indicators. With Recorded Future's vast open source intelligence repository, users can gain deep visibility into the threat landscape by analyzing and visualizing cyber threats.

Recorded Future's Web Intelligence Engine harvests events of many types, including new cyber vulnerabilities and cyber exploits. Just last year, 2014, the world witnessed two major vulnerabilities, *ShellShock*[2] and *Heartbleed*[3] (see Figure 1.1), which both received plenty of media attention. Recorded Future's Web Intelligence Engine picked up more than 250,000 references for these vulnerabilities alone. There are also vulnerabilities that are being openly reported; yet hackers show little or no interest to exploit them.

In this thesis the goal is to use machine learning to examine correlations in vulnerability data from multiple sources, and see if some vulnerability types are more likely to be exploited.

[1] `recordedfuture.com`
[2] `cvedetails.com/cve/CVE-2014-6271`. Retrieved: May 2015
[3] `cvedetails.com/cve/CVE-2014-0160`. Retrieved: May 2015

1.1 Goals

The goals for this thesis are the following:

- Predict which types of cyber vulnerabilities that are turned into exploits and with what probability.
- Find interesting subgroups that are more likely to become exploited.
- Build a binary classifier to predict whether a vulnerability will get exploited or not.
- Give an estimate for a time frame, from a vulnerability being reported until exploited.
- If a good model can be trained, examine how it might be taken into production at Recorded Future.

1.2 Outline

To make the reader acquainted with some of the terms used in the field, this thesis starts with a general description of the cyber vulnerability landscape in Section 1.3. Section 1.4 introduces common data sources for vulnerability data.

Section 2 introduces the reader to machine learning concepts used throughout this thesis. This includes algorithms such as Naive Bayes, Support Vector Machines, k-Nearest-Neighbors, Random Forests, and dimensionality reduction. It also includes information on performance metrics, e.g. precision, recall, and accuracy, and common machine learning workflows such as cross-validation and how to handle unbalanced data.

In Section 3, the methodology is explained. Starting in Section 3.1.1, the information retrieval process from the many vulnerability data sources is explained. Thereafter, there is a short description of some of the programming tools and technologies used. We then present how the vulnerability data can be represented in a mathematical model needed for the machine learning algorithms. The chapter ends with Section 3.5, where we present the data needed for time frame prediction.

Section 4 presents the results. The first part benchmarks different algorithms. In the second part, we explore which features of the vulnerability data that contribute best in classification performance. Third, a final classification analysis is done on the full dataset for an optimized choice of features and parameters. This part includes both binary and probabilistic classification versions. In the fourth part, the results for the exploit time frame prediction are presented.

Finally, Section 5 summarizes the work, and presents conclusions and possible future work.

1.3 The cyber vulnerability landscape

The amount of available information on software vulnerabilities can be described as massive. There is a huge volume of information online about different known vulnerabilities, exploits, proof-of-concept code, etc. A key aspect to keep in mind for this field of cyber vulnerabilities is that open information is not in everyone's best interest. Cyber criminals that can benefit from an exploit will not report it. Whether a vulnerability has been exploited or not may thus be very doubtful. Furthermore, there are vulnerabilities hackers exploit today that the companies do not know exist. If the hacker leaves no or little trace, it may be very hard to see that something is being exploited.

There are mainly two kinds of hackers, the *white hat* and the *black hat*. White hat hackers, such as security experts and researches, will often report vulnerabilities back to the responsible developers. Many major information technology companies, such as Google, Microsoft, and Facebook, award hackers for notifying them about vulnerabilities in so called bug bounty programs[4]. Black hat hackers instead choose to exploit the vulnerability, or sell the information to third parties.

The *attack vector* is the term for the method that a malware or virus is using, typically protocols, services, and interfaces. The *attack surface* of a software environment is the combination of points where different attack vectors can be used. There are several important dates in the vulnerability life cycle:

1. The *creation date* is when the vulnerability is introduced in the software code.
2. The *discovery date* is when a vulnerability is found, either by vendor developers or hackers. The true discovery date is generally not known if a vulnerability is discovered by a black hat hacker.
3. The *disclosure date* is when information about a vulnerability is publicly made available.
4. The *patch date* is when a vendor patch is publicly released.
5. The *exploit date* is when an exploit is created.

Note that the relative order of the three last dates is not fixed; the exploit date may be before or after the patch date and disclosure date. A *zeroday* or *0-day* is an exploit that is using a previously unknown vulnerability. The developers of the code have, when the exploit is reported, had 0 days to fix it, meaning there is no patch available yet. The amount of time it takes for software developers to roll out a fix to the problem varies greatly. Recently, a security researcher discovered a serious flaw in Facebook's image API, allowing him to potentially delete billions of uploaded images (Stockley, 2015). This was reported and fixed by Facebook within 2 hours.

An important aspect of this example is that Facebook could patch this because it controlled and had direct access to update all the instances of the vulnerable software. Contrary, there a many vulnerabilities that have been known for long, have been fixed, but are still being actively exploited. There are cases where the vendor does not have access to patch all the running instances of its software directly, for example Microsoft Internet Explorer (and other Microsoft software). Microsoft (and numerous other vendors) has for long had problems with this, since its software is installed on consumer computers. This relies on the customer actively installing the new security updates and service packs. Many companies are however stuck with old systems which they cannot update for various reasons. Other users with little computer experience are simply unaware of the need to update their software, and left open to an attack when an exploit is found. Using vulnerabilities in Internet Explorer has for long been popular with hackers, both because of the large number of users, but also because there are large amounts of vulnerable users, long after a patch is released.

Microsoft has since 2003 been using *Patch Tuesdays* to roll out packages of security updates to Windows on specific Tuesdays every month (Oliveria, 2005). The following Wednesday is known as *Exploit Wednesday*, since it often takes less than 24 hours for these patches to be exploited by hackers.

Exploits are often used in *exploit kits*, packages automatically targeting a large number of vulnerabilities (Cannell, 2013). Cannell explains that exploit kits are often delivered by an exploit server. For example, a user comes to a web page that has been hacked, but is then immediately forwarded to a malicious server that will figure out the user's browser and OS environment and automatically deliver an exploit kit with a maximum chance of infecting the client with malware. Common targets over the past years have been major browsers, Java, Adobe Reader, and Adobe Flash Player[5]. A whole cyber crime industry has grown around exploit kits, especially in Russia and China (Cannell, 2013). Pre-configured exploit kits are sold as a service for $500 - $10,000 per month. Even if many of the non-premium exploit kits do not use zero-days, they are still very effective since many users are running outdated software.

[4]`facebook.com/whitehat`. Retrieved: May 2015
[5]`cvedetails.com/top-50-products.php`. Retrieved: May 2015

1.4 Vulnerability databases

The data used in this work comes from many different sources. The main reference source is the National Vulnerability Database[6] (NVD), which includes information for all Common Vulnerabilities and Exposures (CVEs). As of May 2015, there are close to 69,000 CVEs in the database. Connected to each CVE is also a list of external references to exploits, bug trackers, vendor pages, etc. Each CVE comes with some Common Vulnerability Scoring System (CVSS) metrics and parameters, which can be found in the Table 1.1. A CVE-number is of the format CVE-Y-N, with a four number year Y, and a 4-6 number identifier N per year. Major vendors are preassigned ranges of CVE-numbers to be registered in the oncoming year, which means that CVE-numbers are not guaranteed to be used or to be registered in consecutive order.

Table 1.1: CVSS Base Metrics, with definitions from Mell et al. (2007).

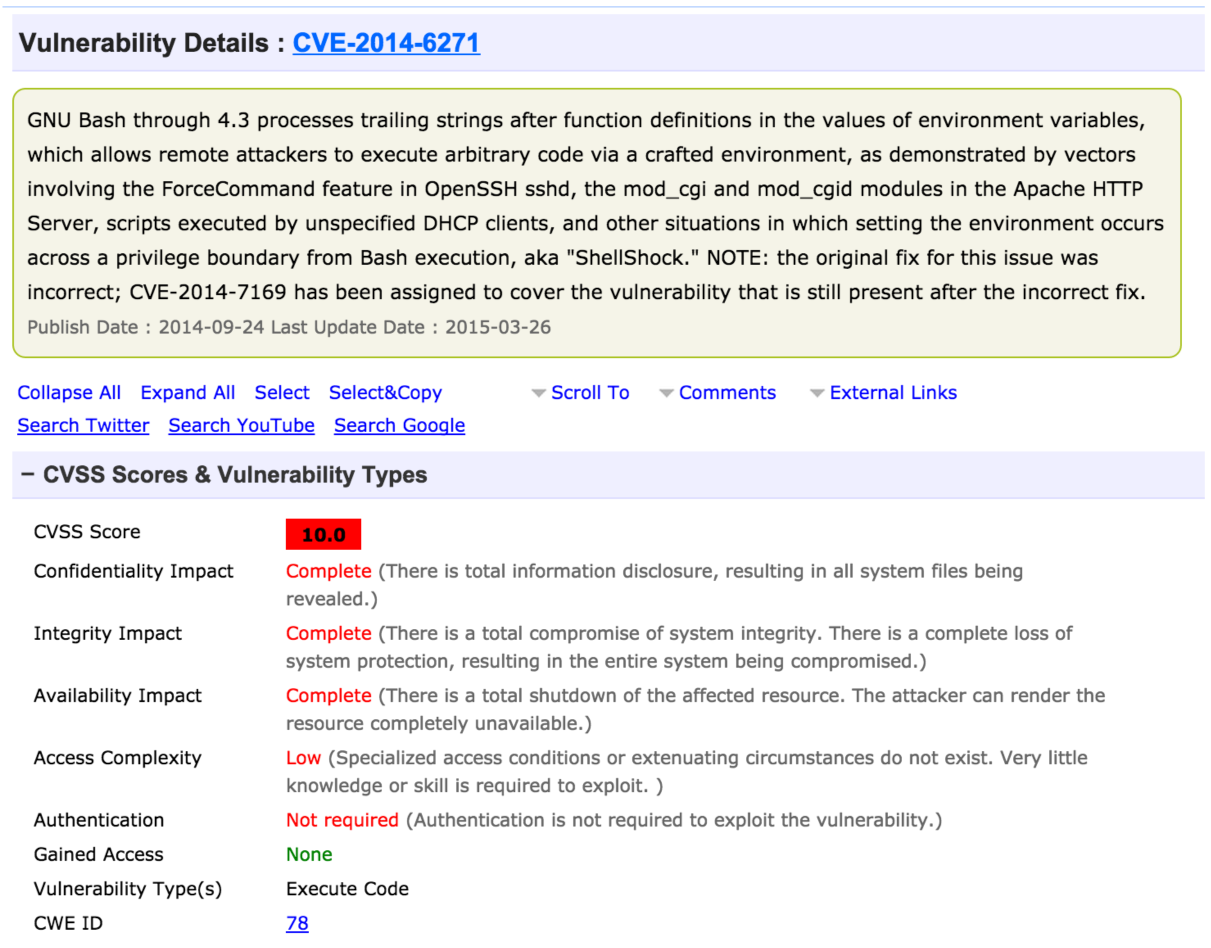
Parameter	**Values**	**Description**
CVSS Score	0-10	This value is calculated based on the next 6 values, with a formula (Mell et al., 2007).
Access Vector	Local Adjacent network Network	The access vector (AV) shows how a vulnerability may be exploited. A local attack requires physical access to the computer or a shell account. A vulnerability with Network access is also called remotely exploitable.
Access Complexity	Low Medium High	The access complexity (AC) categorizes the difficulty to exploit the vulnerability.
Authentication	None Single Multiple	The authentication (Au) categorizes the number of times that an attacker must authenticate to a target to exploit it, but does not measure the difficulty of the authentication process itself.
Confidentiality	None Partial Complete	The confidentiality (C) metric categorizes the impact of the confidentiality, and amount of information access and disclosure. This may include partial or full access to file systems and/or database tables.
Integrity	None Partial Complete	The integrity (I) metric categorizes the impact on the integrity of the exploited system. For example, if the remote attack is able to partially or fully modify information in the exploited system.
Availability	None Partial Complete	The availability (A) metric categorizes the impact on the availability of the target system. Attacks that consume network bandwidth, processor cycles, memory or any other resources affect the availability of a system.
Security Protected	User Admin Other	Allowed privileges
Summary		A free-text description of the vulnerability.

The data from NVD only includes the base CVSS Metric parameters seen in Table 1.1. An example for the vulnerability ShellShock[7] is seen in Figure 1.2. In the official guide for CVSS metrics, Mell et al. (2007) describe that there are similar categories for Temporal Metrics and Environmental Metrics. The first one includes categories for Exploitability, Remediation Level, and Report Confidence. The latter includes Collateral Damage Potential, Target Distribution, and Security Requirements. Part of this data is sensitive and cannot be publicly disclosed.

All major vendors, keep their own vulnerability identifiers as well; Microsoft uses the MS prefix, Red Hat uses RHSA, etc. For this thesis, the work has been limited to study those with CVE-numbers from the NVD.

Linked to the CVEs in the NVD are 1.9M Common Platform Enumeration (CPE) strings, which define different affected software combinations, and contain a software type, vendor, product, and version information. A typical CPE-string could be *cpe:/o:linux: linux_kernel:2.4.1*. A single CVE can affect several

[6] `nvd.nist.gov`. Retrieved: May 2015
[7] `cvedetails.com/cve/CVE-2014-6271`. Retrieved: Mars 2015



Vulnerability Details : CVE-2014-6271

GNU Bash through 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka "ShellShock." NOTE: the original fix for this issue was incorrect; CVE-2014-7169 has been assigned to cover the vulnerability that is still present after the incorrect fix.

Publish Date : 2014-09-24 Last Update Date : 2015-03-26

− CVSS Scores & Vulnerability Types

CVSS Score	10.0
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Execute Code
CWE ID	78

Figure 1.2: An example showing the NVD CVE details parameters for ShellShock.

products, from several different vendors. A bug in the web rendering engine Webkit may affect both of the browsers Google Chrome and Apple Safari, and moreover a range of version numbers.

The NVD also includes 400,000 links to external pages for those CVEs with different types of sources with more information. There are many links to exploit databases such as `exploit-db.com` (EDB), `milw0rm.com`, `rapid7.com/db/modules`, and `1337day.com`. Many links also go to bug trackers, forums, security companies and actors, and other databases. The EDB holds information and proof-of-concept code to 32,000 common exploits, including exploits that do not have official CVE-numbers.

In addition, the NVD includes Common Weakness Enumeration (CWE) numbers[8], which categorizes vulnerabilities into categories such as a *cross-site-scripting* or *OS command injection*. In total there are around 1,000 different CWE-numbers.

Similar to the CWE-numbers there are also Common Attack Patterns Enumeration and Classification numbers (CAPEC). These are not found in the NVD, but can be found in other sources as described in Section 3.1.1.

The general problem within this field is that there is no centralized open source database that keeps all information about vulnerabilities and exploits. The data from the NVD is only partial; for example, within the 400,000 links from NVD, 2,200 are references to the EDB. However, the EDB holds CVE-numbers for every exploit entry, and keeps 16,400 references back to CVE-numbers. This relation is asymmetric since there is a big number of links missing in the NVD. Different actors do their best to cross-reference back to CVE-numbers, and external references. Thousands of these links have over the

[8]The full list of CWE-numbers can be found at `cwe.mitre.org`.

years become dead, since content has been either moved or the actor has disappeared.

The Open Sourced Vulnerability Database[9] (OSVDB) includes far more information than the NVD, for example 7 levels of exploitation (Proof-of-Concept Public, Exploit Public, Exploit Private, Exploit Commercial, Exploit Unknown, Virus / Malware[10], Wormified[10]). There is also information about exploit, disclosure and discovery dates, vendor and third party solutions, and a few other fields. However, this is a commercial database and do not allow open exports for analysis.

In addition, there is a database at CERT (Computer Emergency Response Team) at the Carnegie Mellon University. CERT's database[11] includes vulnerability data, which partially overlaps with the NVD information. CERT Coordination Center is a research center focusing on software bugs and how those impact software and Internet security.

Symantec's Worldwide Intelligence Network Environment (WINE) is more comprehensive and includes data from attack signatures Symantec has acquired through its security products. This data is not openly available. Symantec writes on their webpage[12]:

"WINE is a platform for repeatable experimental research, which provides NSF-supported researchers access to sampled security-related data feeds that are used internally at Symantec Research Labs. Often, today's datasets are insufficient for computer security research. WINE was created to fill this gap by enabling external research on field data collected at Symantec and by promoting rigorous experimental methods. WINE allows researchers to define reference datasets, for validating new algorithms or for conducting empirical studies, and to establish whether the dataset is representative for the current landscape of cyber threats."

1.5 Related work

Plenty of work is being put into research about cyber vulnerabilities by big software vendors, computer security companies, threat intelligence software companies, and independent security researchers. Presented below are some of the findings relevant for this work.

Frei et al. (2006) do a large-scale study of the life-cycle of vulnerabilities, from discovery to patch. Using extensive data mining on commit logs, bug trackers, and mailing lists, they manage to determine the discovery, disclosure, exploit, and patch dates of 14,000 vulnerabilities between 1996 and 2006. This data is then used to plot different dates against each other, for example discovery date vs. disclosure date, exploit date vs. disclosure date, and patch date vs. disclosure date. They also state that black hats create exploits for new vulnerabilities quickly and that the amount of zerodays is increasing rapidly. 90% of the exploits are generally available within a week from disclosure, a great majority within days.

Frei et al. (2009) continue this line of work, and give a more detailed definition of the important dates and describe the main processes of the security ecosystem. They also illustrate the different paths a vulnerability can take from discovery to public, depending on if the discoverer is a black or white hat hacker. The authors provide updated analysis for their plots from Frei et al. (2006). Moreover, they argue that the true discovery dates will never be publicly known for some vulnerabilities since it may be bad publicity for vendors to actually state how long they have been aware of some problem before releasing a patch. Likewise, for a zeroday it is hard or impossible to state how long it has been in use since its discovery. The authors conclude that on average exploit availability has exceeded patch availability since 2000. This gap stresses the need for third party security solutions and need for constantly updated security information of the latest vulnerabilities in order to make patch priority assessments.

Ozment (2007) explains the Vulnerability Cycle, with definitions of the important events of a vulnerability. Those are *Injection Date*, *Release Date*, *Discovery Date*, *Disclosure Date*, *Public Date*, *Patch Date*,

[9] `osvdb.com`. Retrieved: May 2015
[10] Meaning that the exploit has been found in the wild.
[11] `cert.org/download/vul_data_archive`. Retrieved: May 2015
[12] `symantec.com/about/profile/universityresearch/sharing.jsp`. Retrieved: May 2015

Scripting Date. Ozment further argues that the work so far on vulnerability discovery models is using unsound metrics, especially researchers need to consider data dependency.

Massacci and Nguyen (2010) perform a comparative study on public vulnerability databases and compile them into a joint database for their analysis. They also compile a table of recent authors usage of vulnerability databases in similar studies, and categorize these studies as prediction, modeling, or fact finding. Their study focusing on bugs for Mozilla Firefox shows that using different sources can lead to opposite conclusions.

Bozorgi et al. (2010) do an extensive study to predict exploit likelihood using machine learning. The authors use data from several sources such as the NVD, the OSVDB, and the dataset from Frei et al. (2009) to construct a dataset of 93,600 feature dimensions for 14,000 CVEs. The results show a mean prediction accuracy of 90%, both for offline and online learning. Furthermore, the authors predict a possible time frame that an exploit would be exploited within.

Zhang et al. (2011) use several machine learning algorithms to predict time to next vulnerability (TTNV) for various software applications. The authors argue that the quality of the NVD is poor, and are only content with their predictions for a few vendors. Moreover, they suggest multiple explanations of why the algorithms are unsatisfactory. Apart from the missing data, there is also the case that different vendors have different practices for reporting the vulnerability release time. The release time (disclosure or public date) does not always correspond well with the discovery date.

Allodi and Massacci (2012, 2013) study which exploits are actually being used in the wild. They show that only a small subset of vulnerabilities in the NVD, and exploits in the EDB, are found in exploit kits in the wild. The authors use data from Symantec's list of Attack Signatures and Threat Explorer Databases[13]. Many vulnerabilities may have proof-of-concept code in the EDB, but are worthless or impractical to use in an exploit kit. Also, the EDB does not cover all exploits being used in the wild, and there is no perfect overlap between any of the authors' datasets. Just a small percentage of the vulnerabilities are in fact interesting to black hat hackers. The work of Allodi and Massacci is discussed in greater detail in Section 3.3.

Shim et al. (2012) use a game theory model to conclude that "Crime Pays If You Are Just an Average Hacker". In this study they also discuss black markets and reasons why cyber related crime is growing.

Dumitras and Shou (2011) present Symantec's WINE dataset, and hope that this data will be of use for security researchers in the future. As described previously in Section 1.4, this data is far more extensive than the NVD, and includes meta data as well as key parameters about attack patterns, malwares, and actors.

Allodi and Massacci (2015) use the WINE data to perform studies on security economics. They find that most attackers use one exploited vulnerability per software version. Attackers deploy new exploits slowly, and after 3 years about 20% of the exploits are still used.

[13]`symantec.com/security_response/`. Retrieved: May 2015

2 Background

Machine Learning (ML) is a field in computer science focusing on teaching machines to see patterns in data. It is often used to build predictive models, for classification, clustering, ranking, or recommender systems in e-commerce. In supervised machine learning, there is generally some dataset X of observations, with known truth labels y that an algorithm should be trained to predict. $X_{n \times d}$ is called a *feature matrix*, having n samples and d feature dimensions. y, the *target data*, or *label* or *class* vector, is a vector with n elements. y_i denotes the ith observation's *label*, the truth value that a classifier should learn to predict. This is all visualized in figure 2.1. The *feature space* is the mathematical space spanned by the feature dimensions. The *label space* is the space of all possible values to predict.

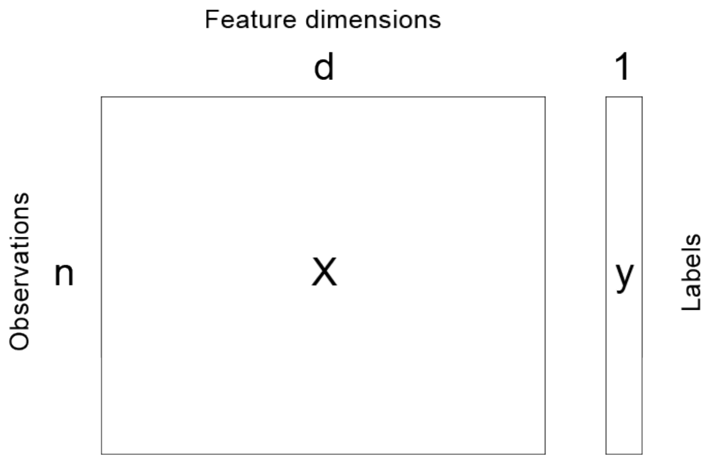


Figure 2.1: A representationof the data setup, with a feature matrix X and a label vector y.

The simplest case of classification is binary classification. For example, based on some weather data for a location, will it rain tomorrow? Based on previously known data X with a boolean label vector y ($y_i \in \{0, 1\}$), a classification algorithm can be taught to see common patterns that lead to rain. This example also shows that each feature dimension may be a completely different space. For example, there is a location in $\mathbb{R}^3$ and maybe 10 binary or categorical features describing some other states, like current temperature, or humidity. It is up to the data scientist to decide which features that are suitable for classification. Feature engineering is a big field of research in itself, and commonly takes up a large fraction of the time spent solving real world machine learning problems.

Binary classification can also be generalized to multi-class classification, where y holds categorical values. The classifier is trained to predict the most likely of several categories. A *probabilistic* classifier does not give each prediction as the most probable class, but returns a probability distribution over all possible classes in the label space. For binary classification this corresponds to predicting the probability of getting rain tomorrow, instead of just answering yes or no.

Classifiers can further be extended to *estimators* (or *regressors*), which will train to predict a value in continuous space. Coming back to the rain example, it would be possible to make predictions on *how much* it will rain tomorrow.

In this section the methods used in this thesis are presented briefly. For an in-depth explanation we refer to the comprehensive works of for example Barber (2012) and Murphy (2012).

2.1 Classification algorithms

2.1.1 Naive Bayes

An example of a simple and fast linear time classifier is the Naive Bayes (NB) classifier. It is rooted in Bayes' theorem, which calculates the probability of seeing the label y given a feature vector x (one row of X). It is a common classifier used in for example spam filtering (Sahami et al., 1998) and document classification, and can often be used as a baseline in classifications.

$$P(y \mid x_1, \ldots, x_d) = \frac{P(y)P(x_1, \ldots x_d \mid y)}{P(x_1, \ldots, x_d)} \tag{2.1}$$

or in plain English

$$\text{probability} = \frac{\text{prior } \cdot \text{ likelihood}}{\text{evidence}} \tag{2.2}$$

The problem is that it is often infeasible to calculate such dependencies, and can require a lot of resources. By using the naive assumption that all x_i are independent,

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d) = P(x_i|y). \tag{2.3}$$

the probability is simplified to a product of independent probabilities

$$P(y \mid x_1, \ldots, x_d) = \frac{P(y) \prod_{i=1}^{d} P(x_i \mid y)}{P(x_1, \ldots, x_d)}. \tag{2.4}$$

The evidence in the denominator is constant with respect to the input x,

$$P(y \mid x_1, \ldots, x_d) \propto P(y) \prod_{i=1}^{d} P(x_i \mid y) \tag{2.5}$$

and is just a scaling factor. To get a classification algorithm, the optimal predicted label $\hat{y}$ should be taken as

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{d} P(x_i \mid y) \tag{2.6}$$

2.1.2 SVM - Support Vector Machines

Support Vector Machines (SVM) is a set of supervised algorithms for classification and regression. Given data points $x_i \in \mathbb{R}^d$, and labels $y_i \in \{-1, 1\}$, for $i = 1, \ldots, n$, a $(d-1)$-dimensional hyperplane $\boldsymbol{w}^T \boldsymbol{x} - b = 0$ is sought to separate two classes. A simple example can be seen in Figure 2.2, where a line (the 1-dimensional hyperplane) is found to separate clusters in $\mathbb{R}^2$. If every hyperplane is forced to go through the origin ($b = 0$), the SVM is said to be *unbiased.*

However, an infinite amount of separating hyperplanes can often be found. More formally, the goal is to find the hyperplane that maximizes the margin between the two classes. This is equivalent to the following optimization problem

$$\min_{\boldsymbol{w}, b} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} \qquad \text{s.t. } \forall i : y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \tag{2.7}$$

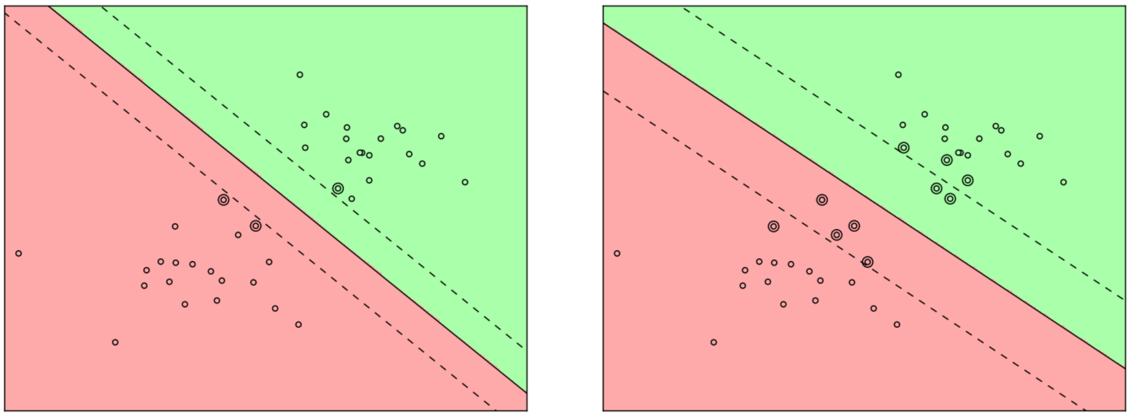


Figure 2.2: For an example dataset an SVM classifier can be used to find a hyperplane, that separates the two classes. The support vectors are the double rounded dots. A lower penalty C in this case (the right image) makes the classifier include more points as support vectors.

2.1.2.1 Primal and dual form

In an easy case, the two classes are linearly separable. It is then possible to find a *hard margin* and a perfect classification. In a real world example, there are often noise and outliers. Forcing a hard margin may overfit the data, and skew the overall results. Instead the constraints can be relaxed using Lagrangian multipliers and the penalty method into a *soft margin* version. Noise and outliers will often make it impossible to satisfy all constraints.

Finding an optimal hyperplane can be formulated as an optimization problem. It is possible to attack this optimization problem from two different angles, the primal and the dual. The primal form is

$$\min_{\boldsymbol{w},b,\zeta} \left\{ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\zeta_i \right\} \qquad \text{s.t. } \forall i : y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \zeta_i \quad \wedge \quad \zeta_i \geq 0 \tag{2.8}$$

By solving the primal problem the optimal weights, $\boldsymbol{w}$, and bias, b, are found. To classify a point, $\boldsymbol{w}^T\boldsymbol{x}$ needs to be calculated. When d is large, the primal problem becomes computationally expensive. Using the duality, the problem can be transformed into a computationally cheaper problem. Since $\alpha_i = 0$ unless x_i is a support vector, the $\boldsymbol{\alpha}$ tends to be a very sparse vector.

The dual problem is

$$\max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^T\boldsymbol{x}_j \right\} \qquad \text{s.t. } \forall i : 0 \leq \alpha_i \leq C \quad \wedge \quad \sum_{i=1}^{n} y_i\alpha_i = 0 \tag{2.9}$$

2.1.2.2 The kernel trick

Since $\boldsymbol{x}_i^T\boldsymbol{x}_j$ in the expressions above is just scalar, it can be replaced by a kernel function $K(x_i, x_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$, which becomes a distance measure between two points. The expressions from above, are called a *linear kernel*, with

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T\boldsymbol{x}_j. \tag{2.10}$$

Specific kernels may be custom tailored to solve specific domain problems. Only standard kernels will be used in this thesis. They are visualized for a toy dataset in Figure 2.3. The Gaussian Radial Basis Function (RBF) is

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2), \qquad \text{for} \qquad \gamma > 0. \tag{2.11}$$

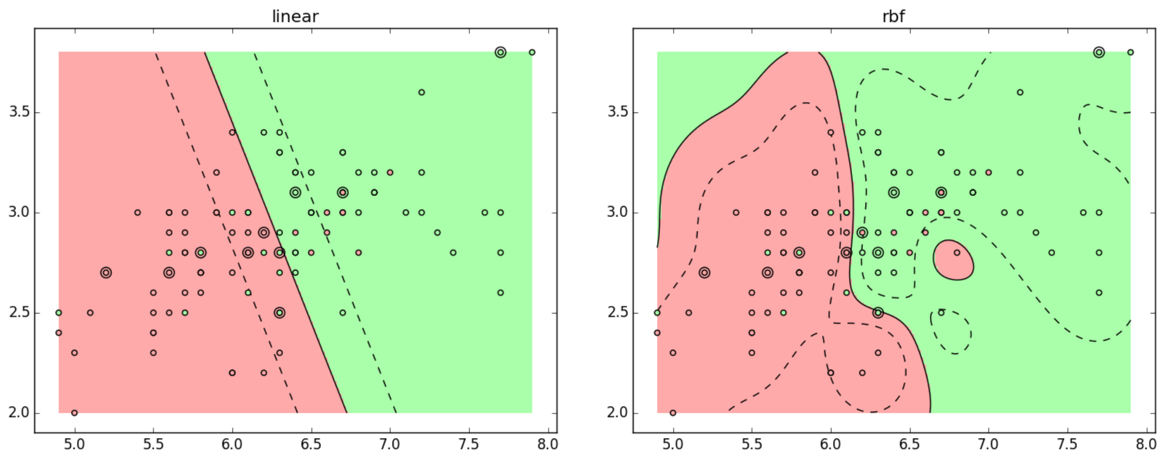


Figure 2.3: This toy example shows a binary classification between red and green points using SVMs with different kernels. An RBF kernel will work much better than a linear kernel by allowing curved boundaries. The test data is drawn in double rounded dots.

2.1.2.3 Usage of SVMs

SVMs can both be used for regression (SVR) and for classification (SVC). SVMs were originally discovered in 1963, and the soft margin version was published 20 years ago (Cortes and Vapnik, 1995).

In the original formulation the goal is to do a binary classification between two classes. SVMs can be generalized to a multi-class version by doing several one-vs-rest classifications, and selecting the class that gets the best separation.

There are many common SVM libraries; two are *LibSVM* and *Liblinear* (Fan et al., 2008). Liblinear is a linear time approximation algorithm for linear kernels, which runs much faster than LibSVM. Liblinear runs in $\mathcal{O}(nd)$, while LibSVM takes at least $\mathcal{O}(n^2d)$.

2.1.3 kNN - k-Nearest-Neighbors

kNN is an algorithm that classifies a new sample x, based on the k nearest neighbors in the feature matrix X. When the k nearest neighbors have been found, it is possible to make several versions of the algorithm. In the standard version, the prediction $\hat{y}$ of the new sample x is taken as the most common class of the k neighbors. In a probabilistic version, the $\hat{y}$ is drawn from a multinomial distribution with class weights equal to the share of the different classes from the k nearest neighbors.

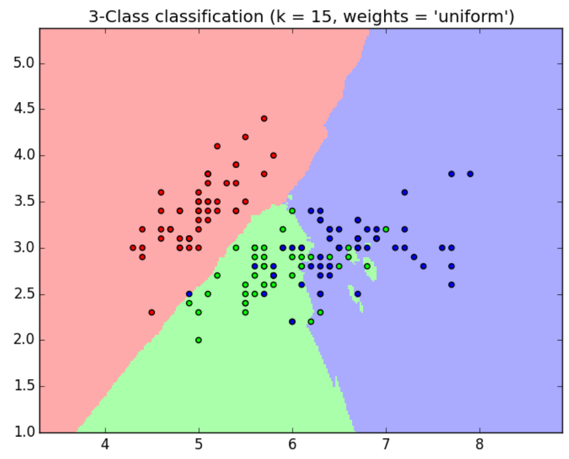
A simple example for kNN can be seen in Figure 2.4, where kNN is used for the Iris dataset[1] using scikit-learn (see Section 3.2).

In the above formulation, *uniform* weights are applied, meaning that all k neighbors are weighted equally. It is also possible to use different *distance* weighting, namely to let the distance from x to the k nearest neighbors weight there relative importance. The distance from a sample x_1 to another sample x_2 is often the Euclidean distance

$$s = \sqrt{\sum_{i=1}^{d}(x_{1i} - x_{2i})^2}, \tag{2.12}$$

but other distance measures are also commonly used. Those include for example the cosine distance, Jaccard similarity, Hamming distance, and the Manhattan distance, which are out of the scope of this thesis. The Euclidean distance measure is not always very helpful for high dimensional data, since all points will lie approximately at the same distance.

[1] `archive.ics.uci.edu/ml/datasets/Iris`. Retrieved: Mars 2015



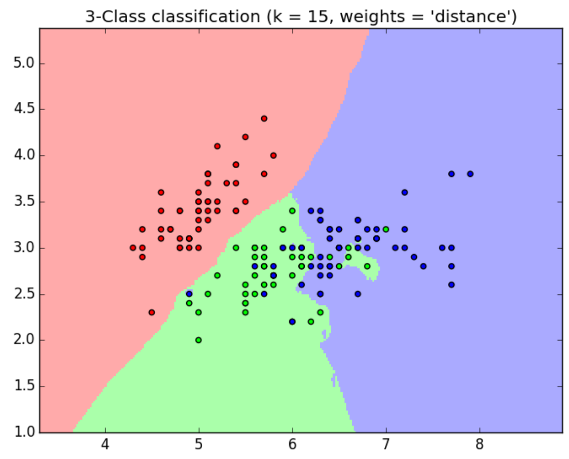


Figure 2.4: In the Iris dataset there are three different flower species, that can be clustered based on same basic measurement. The kNN classifier creates areas in blue, green, and red, for the three species. Any point will be classified to the color of the region. Note that outliers are likely to be misclassified. There is a slight difference between uniform and distance weights as can be seen in the figure, especially how regions around outliers are formed.

kNN differs from SVM (and other model based algorithms) in the sense that all training data is needed for prediction. This makes the algorithm inconvenient for truly big data. The choice of k is problem specific, and needs to be learned and cross-validated. A large k serves as noise-reductions and the algorithm will learn to see larger regions in the data. By using a smaller k the regions get more sensitive, but the algorithm will also learn more noise and overfit.

2.1.4 Decision trees and random forests

A decision tree is an algorithm that builds up a tree of boolean decision rules. A simple example is shown in Figure 2.5[2]. In every node of the tree a boolean decision is made, and the algorithm will then branch until a leaf is reached and a final classification can be made. Decision trees can be used both as classifiers and regressors (as in the figure) to predict some metric.

A decision tree algorithm will build a decision tree from a data matrix X and labels y. It will partition the data, and optimize the boundaries to make the tree as shallow as possible, and make sure that the decisions are made from the most distinct features. A common method for constructing decision trees is the ID3 algorithm (Iterative Dichotomizer 3). How to train a Random Forest classifier will not be covered in this thesis.

Because of the simplicity of the decision trees, they easily overfit to noise and should be used with care. Random Forests is a powerful concept to boost the performance by using an array of decision trees. In the random forest algorithm, the training data is split into random subsets, to make every decision tree in the forest learn differently. In this way, much like raising k in the kNN algorithm, noise is canceled out by learning the same overall structures in many different trees.

2.1.5 Ensemble learning

To get better and more stable results ensemble methods can be used, to aggregate and vote the classification of several independent algorithms. Random Forests is an ensemble learning algorithm, combining many decision trees.

[2] `commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png`. Courtesy of Stephen Milborrow. This image is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

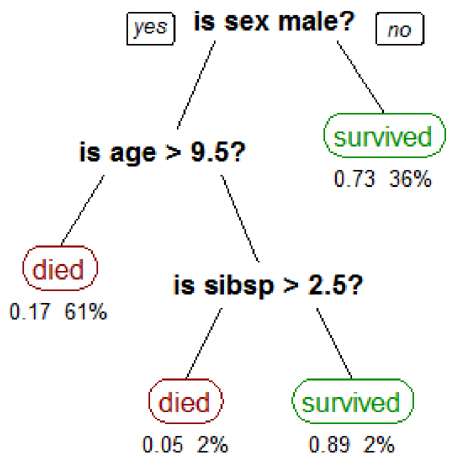


Figure 2.5: A decision tree showing Titanic survivors. The decimal notation shows the probability of survival. The percentage shows the number of cases from the whole dataset. "sibsp" is the number of spouses or siblings aboard for the passenger.

Voting with n algorithms can be applied using

$$\hat{y} = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i}, \tag{2.13}$$

where $\hat{y}$ is the final prediction, y_i and w_i are the prediction and weight of voter i. With uniform weights $w_i = 1$, all voters are equally important. This is for a regression case, but can be transformed to a classifier using

$$\hat{y} = \arg\max_k \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i = k)}{\sum_{i=1}^{n} w_i}, \tag{2.14}$$

taking the prediction to be the class k with the highest support. $\mathbb{I}(x)$ is the indicator function, which return 1 if x is true, else 0.

2.2 Dimensionality reduction

Dimensionally reduction can be a powerful way to reduce a problem from a high dimensional domain to a more simplistic feature set. Given a feature matrix $X_{n \times d}$, find the $k < d$ most prominent axes to project the data onto. This will reduce the size of the data, both noise and information will be lost. Some standard tools for dimensionality reduction include Principal Component Analysis (PCA) (Pearson, 1901) and Fischer Linear Discriminant Analysis (LDA).

For large datasets dimensionality reduction techniques are computationally expensive. As a rescue, it is possible to use approximative and randomized algorithms such as Randomized PCA, and Random Projections, which we present briefly below. We refer to a book in machine learning or statistics for a more comprehensive overview. Dimensionality reduction is not of vital importance for the results of this thesis.

2.2.1 Principal component analysis

For a more comprehensive version of PCA we refer the reader to for example Shlens (2014) or Barber (2012). We assume that the feature matrix X has been centered to have zero mean per column. PCA applies orthogonal transformations to X to find its *principal components.* The first principal component is computed such that the data has largest possible variance in that dimension. The next coming principal components are sequentially computed to be orthogonal to the previous dimensions and have decreasing variance. This can be done be computing the eigenvalues and eigenvectors of the covariance matrix

$$S = \frac{1}{n-1} X X^T. \tag{2.15}$$

The principal components correspond to the orthonormal eigenvectors sorted by the largest eigenvalues. With S symmetric, it is possible to diagonalize it and write it on the form

$$X X^T = W D W^T \tag{2.16}$$

with W being the matrix formed by columns of orthonormal eigenvectors and D being a diagonal matrix with the eigenvalues on the diagonal. It is also possible to use Singular Value Decomposition (SVD) to calculate the principal components. This is often preferred for numerical stability. Let

$$X_{n\times d} = U_{n\times n} \Sigma_{n\times d} V^T_{d\times d}. \tag{2.17}$$

U and V are unitary, meaning $UU^T = U^T U = I$, with I being the identity matrix. Σ is a rectangular diagonal matrix with singular values $\sigma_{ii} \in \mathbb{R}^+_0$ on the diagonal. S can then be written as

$$S = \frac{1}{n-1} X X^T = \frac{1}{n-1} (U\Sigma V^T)(U\Sigma V^T)^T = \frac{1}{n-1} (U\Sigma V^T)(V\Sigma U^T) = \frac{1}{n-1} U\Sigma^2 U^T \tag{2.18}$$

Computing the dimensionality reduction for a large dataset is intractable; for example for PCA the covariance matrix computation is $\mathcal{O}(d^2 n)$, and eigenvalue decomposition is $\mathcal{O}(d^3)$, meaning that a total complexity of $\mathcal{O}(d^2 n + d^3)$.

To make the algorithms faster, randomized approximate versions can be used. Randomized PCA uses Randomized SVD, and several versions are explained by for example Martinsson et al. (2011). This will not be covered further in this thesis.

2.2.2 Random projections

The random projections approach is for the case when $d > n$. Using a random projection matrix $R \in \mathbb{R}^{k\times d}$, a new feature matrix $X' \in \mathbb{R}^{n\times k}$ can be constructed by computing

$$X'_{n\times k} = \frac{1}{\sqrt{k}} X_{n\times d} R_{d\times k}. \tag{2.19}$$

The theory behind random projections builds on the *Johnson-Lindenstrauss lemma* (Johnson and Lindenstrauss, 1984). The lemma states that there is a projection matrix R such that a high dimensional space can be projected onto a lower dimensional space while almost keeping all pairwise distances between any two observations. Using the lemma it is possible to calculate a minimum dimension k to guarantee the overall distortion error.

In a Gaussian random projection the elements of R are drawn from $N(0, 1)$. Achlioptas (2003) shows that it is possible to use a sparse random projection, to speed up the calculations. Li et al. (2006) generalize this and further show that it is possible to achieve an even higher speedup selecting the elements of R as

$$\sqrt{s} \cdot \begin{cases} -1 & & 1/2s \\ 0 & \text{with probability} & 1 - 1/s \\ +1 & & 1/2s \end{cases} \tag{2.20}$$

Achlioptas (2003) uses $s = 1$ or $s = 3$, but Li et al. (2006) show that s can be taken much larger, and recommend using $s = \sqrt{d}$.

2.3 Performance metrics

There are some common performance metrics to evaluate classifiers. For a binary classification there are four cases according to Table 2.1.

Table 2.1: Test case outcomes and definitions. The number of positive and negatives are denoted $P = TP + FP$, and $N = FN + TN$ respectively.

	Condition Positive	Condition Negative
Test Outcome Positive	True Positive (TP)	False Positive (FP)
Test Outcome Negative	False Negative (FN)	True Negative (TN)

Accuracy, precision, and recall are common metrics for performance evaluation.

$$accuracy = \frac{TP + TN}{P + N} \qquad precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \tag{2.21}$$

Accuracy is the share of samples correctly identified. Precision is the share of samples correctly classified as positive. Recall is the share of positive samples classified as positive. Precision and recall and often plotted in so called Precision Recall or Receiver Operating Characteristic (ROC) curves (Davis and Goadrich, 2006). For most real world examples, there are errors and it is hard to achieve a perfect classification. These metrics are used in different environments, and there is always a trade-off. For example, in a hospital, the sensitivity for detecting patients with cancer should be high. It can be considered acceptable to get a large number of false positives, which can be examined more carefully. However, for a doctor to tell a patient that s/he has cancer, the precision should be high.

A probabilistic binary classifier will give a probability estimate $p \in [0, 1]$ for each predicted sample. Using a threshold θ, it is possible to classify the observations. All predictions with $p > \theta$, will be set as class 1, and all other will be class 0. By varying θ, we can achieve an expected performance for precision and recall (Bengio et al., 2005).

Furthermore, to combine both of these measures, the F_β-score is defined as below, often with $\beta = 1$, the F_1*-score*, or just simply *F-score*.

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \qquad F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.22}$$

For probabilistic classification a common performance measure is average logistic loss, cross-entropy loss, or often *logloss* defined as

$$logloss \; = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i \mid \hat{y}_i) \; = -\frac{1}{n}\sum_{i=1}^{n} \Big(y_i \log \hat{y}_i + (y_i - 1)\log(1 - \hat{y}_i)\Big). \tag{2.23}$$

$y_i \in \{0, 1\}$ are binary labels, and $\hat{y}_i = P(y_i = 1)$ is the predicted probability $\hat{y}_i$ that $y_i = 1$.

2.4 Cross-validation

Cross-validation is way to better ensure the performance of a classifier (or an estimator), by performing several data fits and see that a model performs well on unseen data. The available dataset is first shuffled, and then split into two or three parts, for training, testing, and sometimes also validation. The model is for example trained on 80% of the data, and tested on the remaining 20%.

KFold cross-validation is common for evaluating a classifier, and ensure that the model does not overfit. KFold means that the data is split into subgroups X_j for $j = 1, \dots, k$. Over k runs, the classifier is trained on $X \backslash X_j$ and tested on X_j. Commonly $k = 5$ or $k = 10$ are used.

Stratified KFold ensures that the class frequency of y is preserved in every fold y_j. This breaks the randomness to some extent. Stratification can be good for smaller datasets, where truly random selection tend to make the class proportions skew.

2.5 Unequal datasets

In most real world problems the number of observations that can be used to build a machine learning model is not equal between the classes. To continue with the cancer prediction example, there might be lots of examples of patients not having cancer ($y_i = 0$), but only a few patients with positive label ($y_i = 1$). If 10% of the patients have cancer, it is possible to use a naive algorithm that always predicts no cancer, and get 90% accuracy. The recall and F-score would however be zero as described in equation (2.22).

Depending on algorithm, it may be important to use an equal amount of training labels in order not to skew the results (Ganganwar, 2012). One way to handle this is to *subsample* or *undersample* the dataset, to contain an equal amount of each class. This can be done both at random, or with intelligent algorithms trying to remove redundant samples in the majority class. A contrary approach is to *oversample* the minority class(es), simply by duplicating minority observations. Oversampling can also be done in variations, for example by mutating some of the sample points to create artificial samples. The problem with undersampling is that important data is lost, while on the contrary algorithms tend to overfit when oversampling. Additionally, there are hybrid methods trying to mitigate this by mixing under- and oversampling.

Yet another approach is to introduce a more advanced cost function, called *cost sensitive learning.* For example, Akbani et al. (2004) explain how to *weight* the classifier, and adjust the weights inversely proportional to the class frequencies. This will make the classifier penalize harder if samples from small classes are incorrectly classified.

3

Method

In this chapter, we present the general workflow of the thesis. First, the information retrieval process is described, with a presentation of the data used. Then, we describe how this data was analyzed and pre-processed. Third, we build the feature matrix with corresponding labels needed to run the machine learning algorithms. At last, the data needed for predicting exploit time frame is analyzed and discussed.

3.1 Information Retrieval

A large part of this thesis consisted of mining data to build the feature matrix for the ML algorithms. Vulnerability databases were covered in Section 1.4. In this section we first present the open Web data that was mined, and secondly the data from Recorded Future.

3.1.1 Open vulnerability data

To extract the data from the NVD, the GitHub project *cve-search*[1] was used. cve-search is a tool to import CVE, CPE, and CWE data into a MongoDB to facilitate search and processing of CVEs. The main objective of the software is to avoid doing direct and public lookups into the public CVE databases. The project was forked and modified to extract more information from the NVD files.

As mentioned in Section 1.4, the relation between the EDB and the NVD is asymmetrical; the NVD is missing many links to exploits in the EDB. By running a Web scraper script, 16,400 connections between the EDB and CVE-numbers could be extracted from the EDB.

The CAPEC to CVE mappings was scraped of a database using *cve-portal*[2], a tool using the cve-search project. In total, 421,000 CAPEC mappings were extracted, with 112 unique CAPEC-numbers. CAPEC definitions were extracted from an XML file found at `capec.mitre.org`.

The CERT database has many more fields than the NVD, and for some of the vulnerabilities provides more information about fixes and patches, authors, etc. For some vulnerabilities it also includes exploitability and other CVSS assessments not found in the NVD. The extra data is however very incomplete, actually missing for many CVEs. This database also includes date fields (creation, public, first published) that will be examined further when predicting the exploit time frame.

3.1.2 Recorded Future's data

The data presented in Section 3.1.1 was combined with the information found with Recorded Future's Web Intelligence Engine. The engine actively analyses and processes 650,000 open Web sources for

[1]`github.com/wimremes/cve-search`. Retrieved: Jan 2015
[2]`github.com/CIRCL/cve-portal`. Retrieved: April 2015

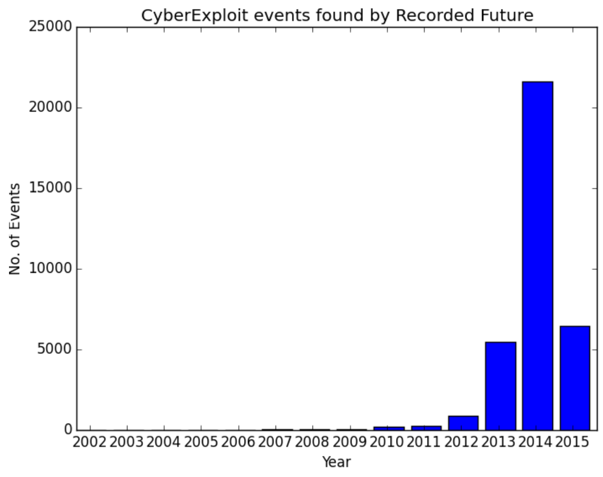


Figure 3.1: **Recorded Future Cyber Exploits**. Number of cyber exploit events per year. The data was retrieved on Mars 23rd, 2015.

information about different types of events. Every day 20M new events are harvested from data sources such as Twitter, Facebook, blogs, news articles, and RSS feeds. This information is made available to customers through an API, and can also be queried and graphically visualized in a Web application. In this work, we use the data of type *CyberExploit*. Every CyberExploit links to a *CyberVulnerability* entity, which is usually a CVE-number, or a vendor number.

The data from Recorded Future is limited since the harvester has mainly been looking at data published in last few years. The distribution in Figure 3.1 shows the increase in activity. There are 7,500 distinct CyberExploit event clusters from open media. These are linked to 33,000 event instances. Recorded Future's engine clusters instances of the same event into clusters with the main event. For example, a tweet with retweets will form multiple event instances, but belong to the same event cluster. These 7,500 main events connect to 700 different vulnerabilities. There is a clear power law distribution, with the most common vulnerabilities having a huge share of the total amount of instances. For example, out of the 33,000 instances 4,700 are for ShellShock (CVE-2014-6271) and 4,200 for Sandworm (CVE-2014-4114). These vulnerabilities are very critical with a huge attack surface, and hence have received large media coverage.

An example of a CyberExploit instance found by Recorded Future can be seen in the JSON blob in Listing 3.1. This is a simplified version; the original data contains much more information. This data comes from a Microsoft blog and contains information about 2 CVEs being exploited in the wild.

Listing 3.1: A simplified JSON Blob example for a detected CyberExploit instance. This shows the detection of a sentence (fragment) containing references to 2 CVE-numbers that are said to have exploits in the wild.

```json
{
  "type": "CyberExploit",
  "start": "2014-10-14T17:05:47.000Z",
  "fragment": "Exploitation of CVE-2014-4148 and CVE-2014-4113 detected in the
      wild.",
  "document": {
     "title": "Accessing risk for the october 2014 security updates",
     "url": "http://blogs.technet.com/b/srd/archive/2014/10/14/accessing-risk-
         for-the-october-2014-security-updates.aspx",
     "language": "eng",
     "published": "2014-10-14T17:05:47.000Z"
  }
}
```

3.2 Programming tools

This section is technical, and is not needed to understand the rest of the thesis. For the machine learning part, the Python project scikit-learn (Pedregosa et al., 2011) was used. scikit-learn is a package of simple and efficient tools for data mining, data analysis, and machine learning. scikit-learn builds upon popular Python packages such as numpy, scipy, and matplotlib[3]. A good introduction to machine learning with Python can be found in (Richert, 2013).

To be able to handle the information in a fast and flexible way, a MySQL[4] database was set up. Python and Scala scripts were used to fetch and process vulnerability data from the various sources. Structuring the data in a relational database also provided a good way to query, pre-process, and aggregate data with simple SQL commands. The data sources could be combined into an aggregated joint table used when building the feature matrix (see Section 3.4). By heavily indexing the tables, data could be queried and grouped much faster.

3.3 Assigning the binary labels

A non trivial part was constructing the label vector y with truth values. There is no parameter to be found in the information from the NVD about whether a vulnerability has been exploited or not. What is to be found are the 400,000 links of which some links to exploit information.

A vulnerability was considered to have an exploit ($y_i = 1$) if it had a link to some known exploit from the CVE-numbers scraped from the EDB. In order not to build the answer y into the feature matrix X, links containing information about exploits were not included. There are three such databases; `exploit-db.com`, `milw0rm.com`, and `rapid7.com/db/modules` (aka metasploit).

As mentioned in several works of Allodi and Massacci (2012, 2013), it is sometimes hard to say whether a vulnerability has been exploited. In one point of view, every vulnerability can be said to be exploiting something. A more sophisticated concept is using different levels of vulnerability exploitation, ranging from simple proof-of-concept code, to a scripted version, to a fully automated version included in some exploit kit. Information like this is available in the OSVDB. According to the work of Allodi and Massacci (2012), the amount of vulnerabilities found in the wild is rather small compared to the total number of vulnerabilities. The EDB also includes many exploits without corresponding CVE-numbers. The authors compile a list of Symantec Attack Signatures into a database called SYM. Moreover, they collect further evidence of exploits used in exploit kits into a database called EKITS. Allodi and Massacci (2013) then write:

1. *The greatest majority of vulnerabilities in the NVD are not included nor in EDB nor in SYM.*
2. *EDB covers SYM for about 25% of its surface, meaning that 75% of vulnerabilities exploited by attackers are never reported in EDB by security researchers. Moreover, 95% of exploits in EDB are not reported as exploited in the wild in SYM.*
3. *Our EKITS dataset overlaps with SYM about 80% of the time.*

A date distribution of vulnerabilities and exploits can be seen in Figure 3.2. This shows that the number of exploits added to the EDB is decreasing. In the same time, the amount of total vulnerabilities added to the NVD is increasing. The blue curve shows the total number of exploits registered in the EDB. The peak for EDB in 2010 can possibly be explained by the fact that the EDB was launched in November 2009[5], after the fall of its predecessor `milw0rm.com`.

Note that an exploit publish date from the EDB often does not represent the *true* exploit date, but the

[3]`scipy.org/getting-started.html`. Retrieved: May 2015
[4]`mysql.com`. Retrieved: May 2015
[5]`secpedia.net/wiki/Exploit-DB`. Retrieved: May 2015

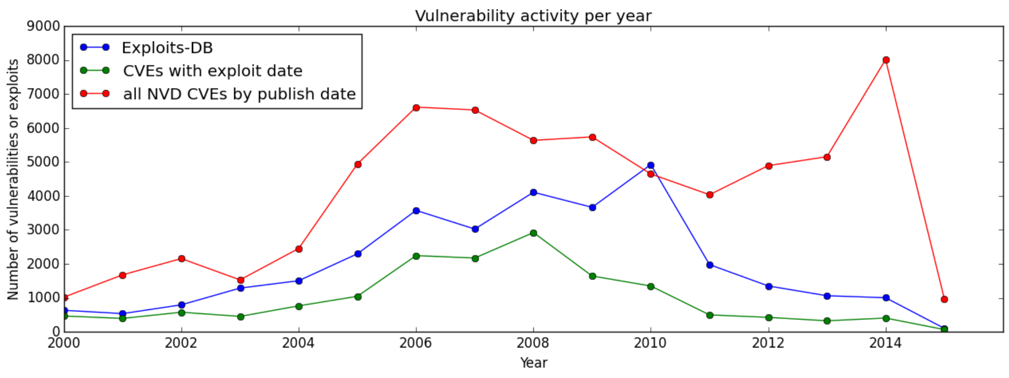


Figure 3.2: Number of vulnerabilities or exploits per year. The red curve shows the total number of vulnerabilities registered in the NVD. The green curve shows the subset of those which have exploit dates from the EDB. The blue curve shows the number of exploits registered in the EDB.

day when an exploit was *published* to the EDB. However, for most vulnerabilities exploits are reported to the EDB quickly. Likewise, a publish date in the NVD CVE database does not always represent the actual disclosure date.

Looking at the decrease in the number of exploits in the EDB, we ask two questions. 1) Is the number of exploits really decreasing? 2) Or are fewer exploits actually reported to the EDB? No matter which is true, there does not seem to be any better open vulnerability sources of whether exploits exist.

The labels extracting from the EDB do not add up in proportions reported the study by Bozorgi et al. (2010), where the dataset consisting of 13,700 CVEs had 70% positive examples.

The result of this thesis would have benefited from gold standard data. The definition for *exploited* in this thesis for the following pages will be if there exists some exploit in the EDB. The *exploit date* will be taken as the exploit publish date in the EDB.

3.4 Building the feature space

Feature engineering is often considered hard, and a lot of time can be spent just on selecting a proper set of features. The data downloaded needs to be converted into a feature matrix, to be used with the ML algorithms. In this matrix, each row is a vulnerability and each column is a feature dimension. A common theorem in the field of machine learning is the No Free Lunch theorem (Wolpert and Macready, 1997), which says that there is no universal algorithm or method that will always work. Thus, to get the proper behavior and results, a lot of different approaches might have to be tried. The feature space is presented in Table 3.1. In the following subsections those features will be explained.

Table 3.1: Feature space. Categorical features are denoted Cat, and converted into one binary dimension per feature. $\mathbb{R}$ denotes real valued dimensions, and in this case all scale in the range 0-10.

Feature group	**Type**	**No. of features**	**Source**
CVSS Score	$\mathbb{R}$	1	NVD
CVSS Parameters	Cat	21	NVD
CWE	Cat	29	NVD
CAPEC	Cat	112	cve-portal
Length of Summary	$\mathbb{R}$	1	NVD
N-grams	Cat	0-20000	NVD
Vendors	Cat	0-20000	NVD
References	Cat	0-1649	NVD
No. of RF CyberExploits	$\mathbb{R}$	1	RF

3.4.1 CVE, CWE, CAPEC, and base features

From Table 1.1 with CVE parameters, categorical features can be constructed. In addition boolean parameters were used for all of the CWE categories. In total 29 distinct CWE-numbers out of 1,000 possible values were found to actually be in use. CAPEC-numbers were also included as boolean parameters, resulting in 112 features.

As suggested by Bozorgi et al. (2010), the length of some fields can be a parameter. The natural logarithm of the length of the NVD summary was used as a base parameter. The natural logarithm of the number of CyberExploit events found by Recorded Future was also used as a feature.

3.4.2 Common words and n-grams

Apart from the structured data, there is unstructured information in the summaries. Using the *CountVectorizer* module from scikit-learn, common words and *n-grams* can be extracted. An n-gram is a tuple of words that occur together in some document. Each CVE summary make up a document, and all the documents make up a *corpus*. Some examples of common n-grams are presented in Table 3.2. Each CVE summary can be vectorized, and converted into a set of occurrence count features, based on which n-grams that summary contains. Note that this is a bag-of-words model on a document level. This is one of the simplest ways to utilize the information found in the documents. There exists far more advanced methods using Natural Language Processing (NLP) algorithms, entity extraction and resolution, etc.

Table 3.2: Common (3,6)-grams found in 28,509 CVEs between 2010-01-01 and 2014-12-31.

n-gram	Count
allows remote attackers	14120
cause denial service	6893
attackers execute arbitrary	5539
execute arbitrary code	4971
remote attackers execute	4763
remote attackers execute arbitrary	4748
attackers cause denial	3983
attackers cause denial service	3982
allows remote attackers execute	3969
allows remote attackers execute arbitrary	3957
attackers execute arbitrary code	3790
remote attackers cause	3646

Table 3.3: Common vendor products from the NVD, using the full dataset.

Vendor	Product	Count
linux	linux kernel	1795
apple	mac os x	1786
microsoft	windows xp	1312
mozilla	firefox	1162
google	chrome	1057
microsoft	windows vista	977
microsoft	windows	964
apple	mac os x server	783
microsoft	windows 7	757
mozilla	seamonkey	695
microsoft	windows server 2008	694
mozilla	thunderbird	662

3.4.3 Vendor product information

In addition to the common words, there is also CPE (Common Platform Enumeration) information available about linked vendor products for each CVE, with version numbers. It is easy to come to the conclusion that some products are more likely to be exploited than others. By adding vendor features, it will be possible to see if some vendors are more likely to have exploited vulnerabilities. In the NVD there are 1.9M vendor products, which means that there are roughly 28 products per vulnerability. However, for some products, specific versions are listed very carefully. For example, CVE-2003-0001[6] lists 20 different Linux Kernels with versions between 2.4.1-2.4.20, 15 versions of Windows 2000, and 5 versions of NetBSD. To make more sense of this data, the 1.9M entries were queried for distinct combinations of vendor products per CVE, ignoring version numbers. This led to the result in Table 3.3. Note that the product names and versions are inconsistent (especially for Windows) and depend on who reported the CVE. No effort was put into fixing this.

[6] `cvedetails.com/cve/CVE-2003-0001`. Retrieved: May 2015

In total, the 1.9M rows were reduced to 30,000 different combinations of CVE-number, vendor, and product. About 20,000 of those were unique vendor products that only exist for a single CVE. Some 1,000 of those has vendor products with 10 or more CVEs.

When building the feature matrix, boolean features were formed for a large number of common vendor products. For CVE-2003-0001, that led to three different features being activated, namely "microsoft:windows_2000", "netbsd:netbsd", and "linux:linux_kernel".

3.4.4 External references

The 400,000 external links, were also used as features using extracted domain names. In Table 3.4 the most common references are shown. To filter out noise, only domains occurring 5 or more times were used as features.

Table 3.4: Common references from the NVD. These are for CVEs for the years of 2005-2014.

Domain	**Count**	**Domain**	**Count**
securityfocus.com	32913	debian.org	4334
secunia.com	28734	lists.opensuse.org	4200
xforce.iss.net	25578	openwall.com	3781
vupen.com	15957	mandriva.com	3779
osvdb.org	14317	kb.cert.org	3617
securitytracker.com	11679	us-cert.gov	3375
securityreason.com	6149	redhat.com	3299
milw0rm.com	6056	ubuntu.com	3114

3.5 Predict exploit time-frame

A security manager is not only interested in a binary classification on whether an exploit is likely to occur at some point in the future. Prioritizing which software to patch first is vital, meaning the security manager is also likely to wonder about how soon an exploit is likely to happen if at all.

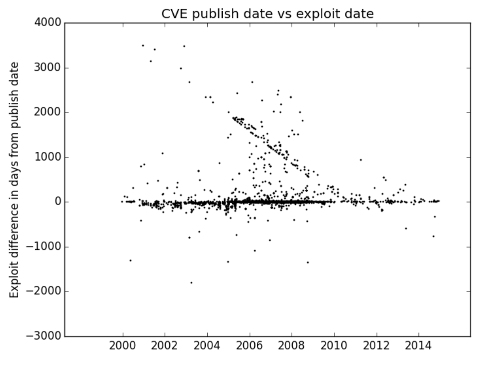


Figure 3.3: Scatter plot of vulnerability publish date vs. exploit date. This plot confirms the hypothesis that there is something misleading about the exploit date. Points below 0 on the y-axis are to be considered zerodays.

A first look at the date distribution of vulnerabilities and exploits was seen in Figure 3.2 and discussed in Section 3.3. As mentioned the dates in the data do not necessarily represent the true dates. In Figure 3.3 exploit publish dates from the EDB are plotted against CVE publish dates. There is a large number of zeroday vulnerabilities (at least they have exploit dates before the CVE publish date). Since these are the only dates available, they will be used as truth values for the machine learning algorithms to follow, even though their correctness cannot be guaranteed. In several studies (Bozorgi et al., 2010, Frei et al., 2006, 2009) the disclosure dates and exploit dates were known to much better extent.

There are certain CVE publish dates that are very frequent. Those form vertical lines of dots. A more detailed analysis shows that there are for example 1098 CVEs published on 2014-12-31. These might be dates when major vendors are given ranges of CVE-numbers to be filled in later. Apart from the NVD publish dates there are also dates to be found the in CERT database. In Figure 3.4 the correlation between NVD publish dates and CVE public dates are shown. This shows that there is a big difference for some dates. Note that all CVEs do not have public date in the CERT database. Therefore, CERT public dates will also be used as a separate case.

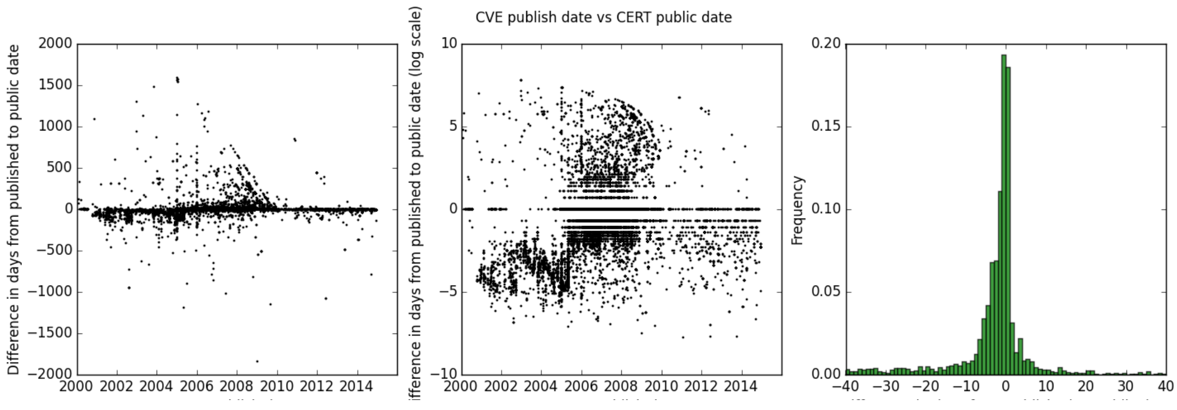


Figure 3.4: NVD published date vs. CERT public date.

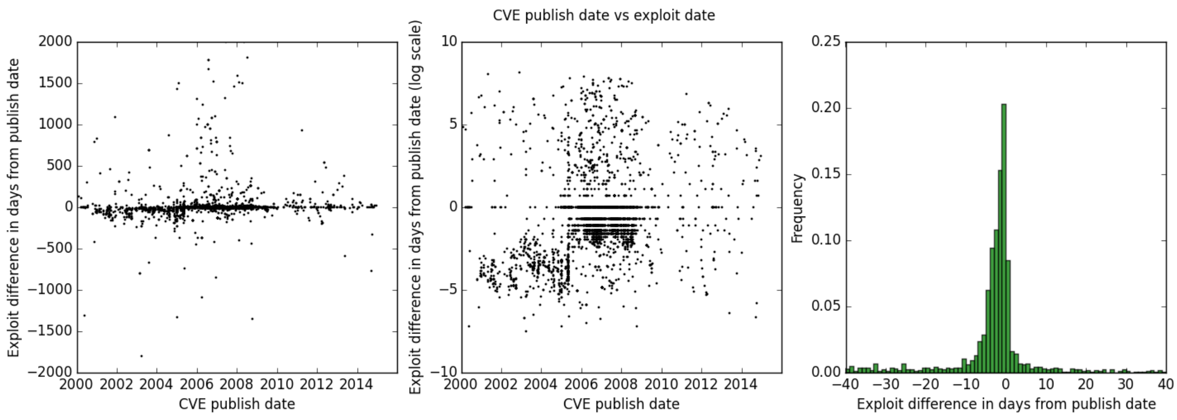


Figure 3.5: NVD publish date vs. exploit date.

Looking at the data in Figure 3.3, it is possible to get more evidence for the hypothesis that there is something misleading with the exploit dates from 2010. There is a line of scattered points with exploit dates in 2010, but with CVE publish dates before. The exploit dates are likely to be invalid, and those CVEs will be discarded. In Figures 3.5 and 3.6, vulnerabilities with exploit dates in 2010, and publish dates more than a 100 days prior were filtered out to remove points on this line. These plots scatter NVD publish dates vs. exploit dates, and CERT public dates vs. exploit dates respectively. We see that most vulnerabilities have exploit dates very close to the public or publish dates, and there are more zerodays. The time to exploit closely resembles the findings of Frei et al. (2006, 2009), where the authors show that 90% of the exploits are available within a week after disclosure.

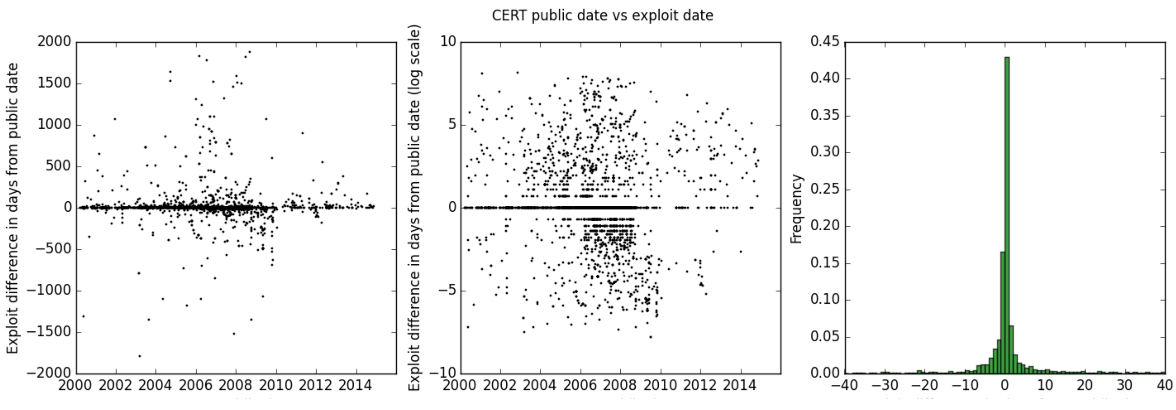


Figure 3.6: CERT public date vs. exploit date.

4
Results

For the binary classification the challenge is to find some algorithm and feature space that give an optimal classification performance. A number of different methods described in Chapter 2 were tried for different feature spaces. The number of possible combinations to construct feature spaces is enormous. Combine that with all possible combinations of algorithms with different parameters and the number of test cases is even more infeasible. The approach to solve this problem was first to fix a feature matrix to run the different algorithms on and benchmark the performance. Each algorithm has a set of *hyper parameters* that need to be tuned for the dataset; for example the penalty factor, C, for SVM, or the number of neighbors, k, for kNN. The number of vendor products used will be denoted n_v, the number of words or n-grams n_w, and the number of references n_r.

Secondly, some methods performing well on average were selected to be tried out on different kinds of feature spaces, to see what kind of features that give the best classification. After that, a feature set was fixed for final benchmark runs where a final classification result could be determined.

The results for the date prediction are presented in Section 4.6. Here, we use the same dataset as in the final binary classification, but with multi-class labels to predict whether an exploit is likely to happen within a day, a week, a month, or a year.

All calculations were carried out on a 2014 MacBook Pro, using 16GB RAM and a dual core 2.6 GHz Intel Core i5 with 4 logical cores. In Table 4.1 all the algorithms used are listed, including their implementation in scikit-learn.

Table 4.1: **Algorithm Implementation.** The following algorithms are used with their default settings, if nothing else is stated.

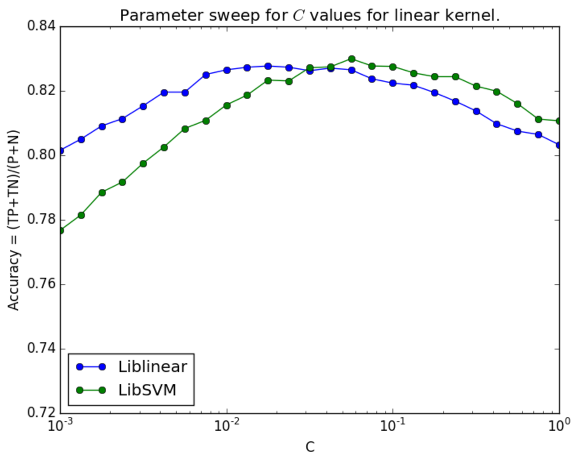
Algorithm	**Implementation**
Naive Bayes	*sklearn.naive_bayes.MultinomialNB()*
SVC, Liblinear	*sklearn.svm.LinearSVC()*
SVC, linear LibSVM	*sklearn.svm.SVC(kernel='linear')*
SVC, RBF LibSVM	*sklearn.svm.SVC(kernel='rbf')*
Random Forest	*sklearn.ensemble.RandomForestClassifier()*
kNN	*sklearn.neighbors.KNeighborsClassifier()*
Dummy	*sklearn.dummy.DummyClassifier()*
Randomized PCA	*sklearn.decomposition.RandomizedPCA()*
Random Projections	*sklearn.random_projection.SparseRandomProjection()*

4.1 Algorithm benchmark

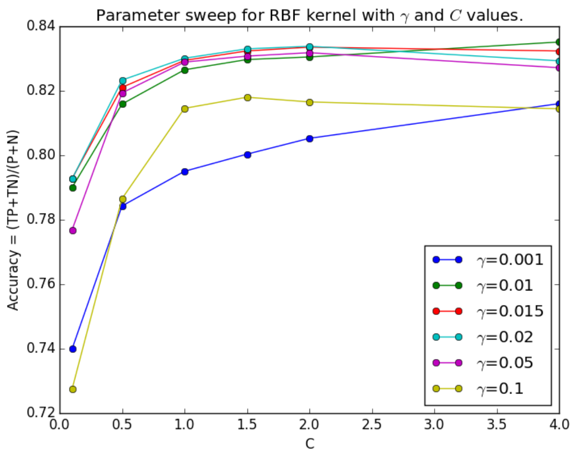
As a first benchmark, the different algorithms described in Chapter 2 were tuned on a dataset consisting of data from 2010-01-01 - 2014-12-31 containing 7,528 samples, with an equal amount of exploited and unexploited CVEs. A basic feature matrix was constructed with the base CVSS parameters, CWE-numbers, $n_v = 1000$, $n_w = 1000$ ((1,1)-grams), and $n_r = 1000$. The results are presented in Figure 4.1

with the best instances summarized in Table 4.2. Each data point in the figures is the average of a 5-Fold cross-validation.

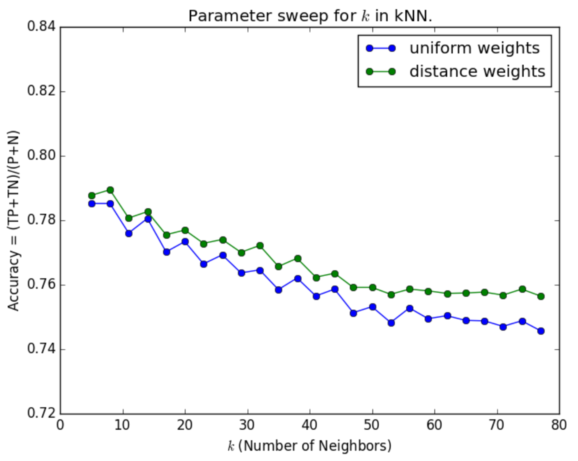
In Figure 4.1a there are two linear SVM kernels. *LinearSVC* in scikit-learn uses *Liblinear* and runs in $\mathcal{O}(nd)$, while *SVC* with linear kernel uses *LibSVM* and takes at least $\mathcal{O}(n^2d)$.



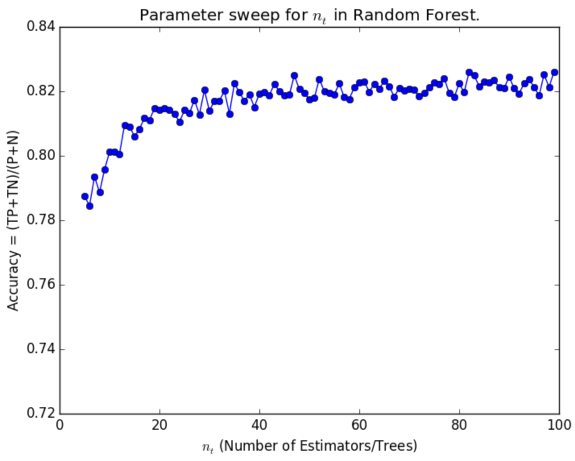
(a) SVC with linear kernel



(b) SVC with RBF kernel



(c) kNN



(d) Random Forest

Figure 4.1: **Benchmark: Algorithms**. Choosing the correct hyper parameters is important; the performance of different algorithm vary greatly when their parameters are tuned. SVC with linear (4.1a) or RBF kernels (4.1b), and Random Forests (4.1d) yield a similar performance, although LibSVM is much faster. kNN (4.1c) is not competitive on this dataset, since the algorithm will overfit if a small k is chosen.

Table 4.2: **Summary of Benchmark: Algorithms**. Summary of the best scores in Figure 4.1.

Algorithm	**Parameters**	**Accuracy**
Liblinear	$C = 0.023$	0.8231
LibSVM linear	$C = 0.1$	0.8247
LibSVM RBF	$\gamma = 0.015, C = 4$	0.8368
kNN	$k = 8$	0.7894
Random Forest	$n_t = 99$	0.8261

A set of algorithms was then constructed using these parameters and cross-validated on the entire dataset again. This comparison, seen in Table 4.3 also included the Naive Bayes Multinomial algorithm, which does not have any tunable parameters. The LibSVM and Liblinear linear kernel algorithms were taken with their best values. Random forest was taken with $n_t = 80$, since average results do not improve much after that. Trying $n_t = 1000$ would be computationally expensive, and better methods exist. kNN does not improve at all with more neighbors, and overfits with too few. For the oncoming tests, some of the algorithms will be discarded. In general, Liblinear seems to perform best based on speed and classification accuracy.

Table 4.3: **Benchmark: Algorithms.** Comparison of the different algorithms with optimized hyper parameters. The results are the mean of 5-Fold cross-validation. Standard deviations are all below 0.01 and are omitted.

Algorithm	Parameters	**Accuracy**	**Precision**	**Recall**	**F_1**	**Time**
Naive Bayes		0.8096	0.8082	0.8118	0.8099	0.15s
Liblinear	$C = 0.02$	0.8466	0.8486	0.8440	0.8462	0.45s
LibSVM linear	$C = 0.1$	0.8466	0.8461	0.8474	0.8467	16.76s
LibSVM RBF	$C = 2, \gamma = 0.015$	0.8547	0.8590	0.8488	0.8539	22.77s
kNN, distance	$k = 80$	0.7667	0.7267	0.8546	0.7855	2.75s
Random Forest	$n_t = 80$	0.8366	0.8464	0.8228	0.8344	31.02s

4.2 Selecting a good feature space

By fixing a subset of algorithms it is possible to further investigate the importance of different features. To do this some of the features were removed to better see how different features contribute and scale. For this the algorithms SVM Liblinear and Naive Bayes were used. Liblinear performed well in the initial benchmark, both in terms of accuracy and speed. Naive Bayes is even faster, and will be used as a baseline in some of the following benchmarks. For all the plots and tables to follow in this section, each data point is the result of a 5-Fold cross-validation for that dataset and algorithm instance.

In this section we will also list a number of tables (Table 4.4, Table 4.5, Table 4.7, Table 4.8, and Table 4.9) with naive probabilities for exploit likelihood for different features. **Unexploited** and **Exploited** denote the count of vulnerabilities, strictly $|\{y_i \in y \mid y_i = c\}|$ for the two labels, $c \in \{0, 1\}$. The tables were constructed from 55,914 vulnerabilities from 2005-01-01 to 2014-12-31. Out of those 40,833 have no known exploits, while 15,081 do. The column **Prob.** in the tables denotes the naive probability for that feature to lead to an exploit. These numbers should serve as general markers to features with high exploit probability, but not be used as an absolute truth. In the Naive Bayes algorithm, these numbers are combined to infer exploit likelihood. The tables only shown features that are good indicators for exploits, but there exists many other features that will be indicators for when no exploit is to be predicted. For the other benchmark tables in this section, precision, recall, and F-score will be taken for the positive class.

4.2.1 CWE- and CAPEC-numbers

CWE-numbers can be used as features, with naive probabilities shown in Table 4.4. This shows that there are some CWE-numbers that are very likely to be exploited. For example CWE id 89, SQL injection, has 3855 instances, and 2819 of those are associated with exploited CVEs. This makes the total probability of exploit:

$$P = \frac{\frac{2819}{15081}}{\frac{2819}{15081} + \frac{1036}{40833}} \approx 0.8805 \qquad (4.1)$$

Like above a probability table is shown for the different CAPEC-numbers in Table 4.5. There are just a few CAPEC-numbers that have high probability. A benchmark was setup and the results are shown in Table 4.6. The results show that CAPEC information is marginally useful even if the amount of information is extremely poor. The overlap with the CWE-numbers is big since both of these represent the same category of information. Each CAPEC-number has a list of corresponding CWE-numbers.

Table 4.4: **Probabilities for different CWE-numbers being associated with an exploited vulnerability**. The table is limited to references with 10 or more observations in support.

CWE	Prob.	Cnt	Unexp.	Expl.	Description
89	0.8805	3855	1036	2819	Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')
22	0.8245	1622	593	1029	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
94	0.7149	1955	1015	940	Failure to Control Generation of Code ('Code Injection')
77	0.6592	12	7	5	Improper Sanitization of Special Elements used in a Command ('Command Injection')
78	0.5527	150	103	47	Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')
134	0.5456	153	106	47	Uncontrolled Format String
79	0.5115	5340	3851	1489	Failure to Preserve Web Page Structure ('Cross-site Scripting')
287	0.4860	904	670	234	Improper Authentication
352	0.4514	828	635	193	Cross-Site Request Forgery (CSRF)
119	0.4469	5139	3958	1181	Failure to Constrain Operations within the Bounds of a Memory Buffer
19	0.3845	16	13	3	Data Handling
20	0.3777	3064	2503	561	Improper Input Validation
264	0.3325	3623	3060	563	Permissions, Privileges, and Access Controls
189	0.3062	1163	1000	163	Numeric Errors
255	0.2870	479	417	62	Credentials Management
399	0.2776	2205	1931	274	Resource Management Errors
16	0.2487	257	229	28	Configuration
200	0.2261	1704	1538	166	Information Exposure
17	0.2218	21	19	2	Code
362	0.2068	296	270	26	Race Condition
59	0.1667	378	352	26	Improper Link Resolution Before File Access ('Link Following')
310	0.0503	2085	2045	40	Cryptographic Issues
284	0.0000	18	18	0	Access Control (Authorization) Issues

Table 4.5: **Probabilities for different CAPEC-numbers being associated with an exploited vulnerability**. The table is limited to CAPEC-numbers with 10 or more observations in support.

CAPEC	Prob.	Cnt	Unexp.	Expl.	Description
470	0.8805	3855	1036	2819	Expanding Control over the Operating System from the Database
213	0.8245	1622	593	1029	Directory Traversal
23	0.8235	1634	600	1034	File System Function Injection, Content Based
66	0.7211	6921	3540	3381	SQL Injection
7	0.7211	6921	3540	3381	Blind SQL Injection
109	0.7211	6919	3539	3380	Object Relational Mapping Injection
110	0.7211	6919	3539	3380	SQL Injection through SOAP Parameter Tampering
108	0.7181	7071	3643	3428	Command Line Execution through SQL Injection
77	0.7149	1955	1015	940	Manipulating User-Controlled Variables
139	0.5817	4686	3096	1590	Relative Path Traversal

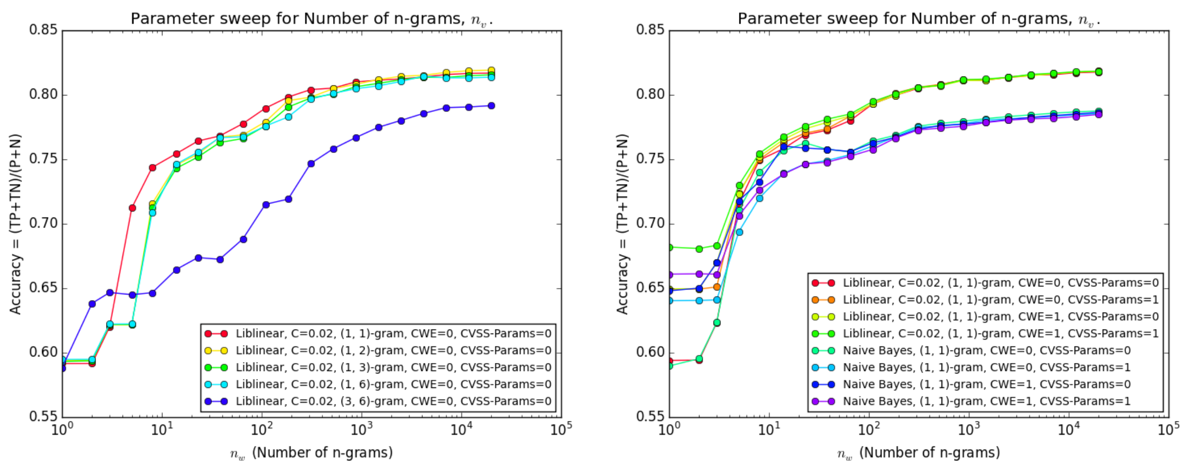
4.2.2 Selecting amount of common n-grams

By adding common words and n-grams as features it is possible to discover patterns that are not captured by the simple CVSS and CWE parameters from the NVD. A comparison benchmark was setup and the results are shown in Figure 4.2. As more common n-grams are added as features, the classification gets better. Note that (1,1)-grams are words. Calculating the most common n-grams is an expensive part when building the feature matrix. In total, the 20,000 most common n-grams were calculated and used

Table 4.6: **Benchmark: CWE and CAPEC**. Using SVM Liblinear linear kernel with $C = 0.02$, $n_v = 0$, $n_w = 0$, $n_r = 0$. Precision, Recall, and F-score are for detecting the exploited label.

CAPEC	**CWE**	**Accuracy**	**Precision**	**Recall**	$\mathbf{F_1}$
No	No	0.6272	0.6387	0.5891	0.6127
No	Yes	0.6745	0.6874	0.6420	0.6639
Yes	No	0.6693	0.6793	0.6433	0.6608
Yes	Yes	0.6748	0.6883	0.6409	0.6637

as features, like those seen in Table 3.2. Figure 4.2 also show that the CVSS and CWE parameters are redundant if there are many n-grams.



(a) N-gram sweep for multiple versions of n-grams, with base information disabled.

(b) N-gram sweep, with and without the base CVSS features and CWE-numbers.

Figure 4.2: **Benchmark: N-grams (words)**. Parameter sweep for n-gram dependency.

In Figure 4.2a it is shown that increasing k in the $(1, k)$-grams, give small differences, but does not yield better performance. This is due to the fact that for all more complicated n-grams, the words they are combined from are already represented as standalone features. As an example, the frequency of the 2-gram (remote, attack) is never more frequent than its base words. However, (j, k)-grams with $j > 1$ get much lower accuracy. Frequent single words are thus important for the classification.

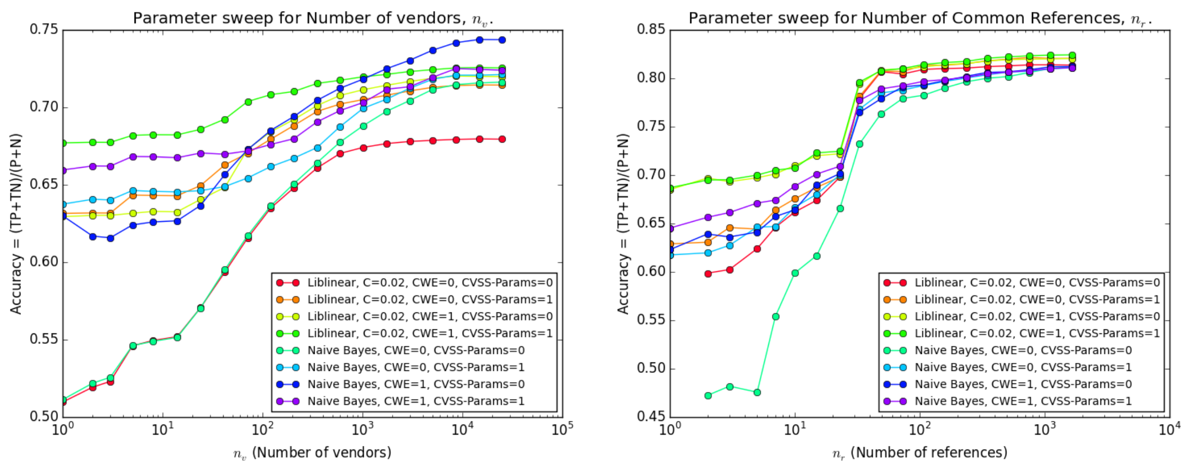
In Figure 4.2b the classification with just the base information performs poorly. By using just a few common words from the summaries it is possible to boost the accuracy significantly. Also for the case when there is no base information, the accuracy gets better with just a few common n-grams. However, using many common words requires a huge document corpus to be efficient. The risk of overfitting the classifier gets higher as n-grams with less statistical support are used. As a comparison the Naive Bayes classifier is used. There are clearly some words that occur more frequently in exploited vulnerabilities. In Table 4.7 some probabilities of the most distinguishable common words are shown.

4.2.3 Selecting amount of vendor products

A benchmark was setup to investigate how the amount of available vendor information matters. In this case, each vendor product is added as a feature. The vendor products are added in the order of Table 3.3, with most significant first. The naive probabilities are presented in Table 4.8 and are very different from the words. In general there are just a few vendor products, like the CMS system *Joomla*, that are very significant. Results are shown in Figure 4.3a. Adding more vendors gives a better classification, but the best results are some 10%-points worse than when n-grams are used in Figure 4.2. When no CWE or base CVSS parameters are used the classification is based solely on vendor products. When a lot of

Table 4.7: **Probabilities for different words being associated with an exploited vulnerability**. The table is limited to words with more 20 observations in support.

Word	Prob.	Count	Unexploited	Exploited
aj	0.9878	31	1	30
softbiz	0.9713	27	2	25
classified	0.9619	31	3	28
m3u	0.9499	88	11	77
arcade	0.9442	29	4	25
root_path	0.9438	36	5	31
phpbb_root_path	0.9355	89	14	75
cat_id	0.9321	91	15	76
viewcat	0.9312	36	6	30
cid	0.9296	141	24	117
playlist	0.9257	112	20	92
forumid	0.9257	28	5	23
catid	0.9232	136	25	111
register_globals	0.9202	410	78	332
magic_quotes_gpc	0.9191	369	71	298
auction	0.9133	44	9	35
yourfreeworld	0.9121	29	6	23
estate	0.9108	62	13	49
itemid	0.9103	57	12	45
category_id	0.9096	33	7	26



(a) **Benchmark: Vendors Products**.

(b) **Benchmark: External References**.

Figure 4.3: Hyper parameter sweep for vendor products and external references. See Sections 4.2.3 and 4.2.4 for full explanations.

vendor information is used, the other parameters matter less, and the classification gets better. But the classifiers also risk overfitting and getting discriminative since many smaller vendor products only have a single vulnerability. Again, Naive Bayes is run against the Liblinear SVC classifier.

4.2.4 Selecting amount of references

A benchmark was also setup to test how common references can be used as features. The references are added in the order of Table 3.4, with most frequent added first. There are in total 1649 references used, with 5 or more occurrences in the NVD. The results for the references sweep is shown in Figure 4.3b. Naive probabilities are presented in Table 4.9. As described in Section 3.3, all references linking

Table 4.8: **Probabilities for different vendor products being associated with an exploited vulnerability**. The table is limited to vendor products with 20 or more observations in support. Note that these features are not as descriptive as the words above. The product names have not been cleaned from the CPE descriptions, and contain many strange names such as *joomla%21*.

Vendor Product	**Prob.**	**Count**	**Unexploited**	**Exploited**
joomla%21	0.8931	384	94	290
bitweaver	0.8713	21	6	15
php-fusion	0.8680	24	7	17
mambo	0.8590	39	12	27
hosting_controller	0.8575	29	9	20
webspell	0.8442	21	7	14
joomla	0.8380	227	78	149
deluxebb	0.8159	29	11	18
cpanel	0.7933	29	12	17
runcms	0.7895	31	13	18

Table 4.9: **Probabilities for different references being associated with an exploited vulnerability**. The table is limited to references with 20 or more observations in support.

References	**Prob.**	**Count**	**Unexploited**	**Exploited**
downloads.securityfocus.com	1.0000	123	0	123
exploitlabs.com	1.0000	22	0	22
packetstorm.linuxsecurity.com	0.9870	87	3	84
shinnai.altervista.org	0.9770	50	3	47
moaxb.blogspot.com	0.9632	32	3	29
advisories.echo.or.id	0.9510	49	6	43
packetstormsecurity.org	0.9370	1298	200	1098
zeroscience.mk	0.9197	68	13	55
browserfun.blogspot.com	0.8855	27	7	20
sitewat.ch	0.8713	21	6	15
bugreport.ir	0.8713	77	22	55
retrogod.altervista.org	0.8687	155	45	110
nukedx.com	0.8599	49	15	34
coresecurity.com	0.8441	180	60	120
security-assessment.com	0.8186	24	9	15
securenetwork.it	0.8148	21	8	13
dsecrg.com	0.8059	38	15	23
digitalmunition.com	0.8059	38	15	23
digitrustgroup.com	0.8024	20	8	12
s-a-p.ca	0.7932	29	12	17

to exploit databases were removed. Some references are clear indicators that an exploit exists, and by using only references it is possible to get a classification accuracy above 80%. The results for this section show the same pattern as when adding more vendor products, or words; when more features are added the classification accuracy quickly gets better over the first hundreds of features and is then gradually saturated.

4.3 Final binary classification

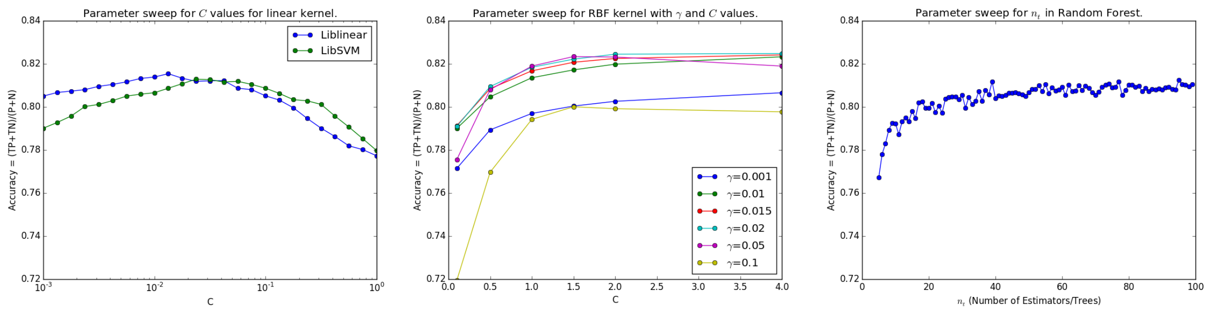
A final run was done with a larger dataset, using 55,914 CVEs between 2005-01-01 - 2014-12-31. For this round the feature matrix was fixed to $n_v = 6000, n_w = 10000, n_r = 1649$. No RF or CAPEC data was used. CWE and CVSS base parameters were used. The dataset was split into two parts (see Table 4.10); one SMALL set consisting of 30% of the exploited CVEs and an equal amount of unexploited; and the rest of the data in the LARGE set.

Table 4.10: **Final Dataset.** Two different datasets were used for the final round. SMALL was used to search for optimal hyper parameters. LARGE was used for the final classification benchmark.

Dataset	Rows	Unexploited	Exploited
SMALL	9048	4524	4524
LARGE	46866	36309	10557
Total	55914	40833	15081

4.3.1 Optimal hyper parameters

The SMALL set was used to learn the optimal hyper parameters for the algorithms on isolated data, like in Section 4.1. Data that will be used for validation later should not be used when searching for optimized hyper parameters. The optimal hyper parameters were then tested on the LARGE dataset. Sweeps for optimal hyper parameters are shown in Figure 4.4.



(a) SVC with linear kernel (b) SVC with RBF kernel (c) Random Forest

Figure 4.4: **Benchmark: Algorithms Final.** The results here are very similar to the results found in the initial discovery run in shown in Figure 4.1. Each point is a calculation based on 5-Fold cross-validation.

A summary of the parameter sweep benchmark with the best results can be seen in Table 4.11. The optimal parameters are slightly changed from the initial data in Section 4.1, but show the overall same behaviors. Accuracy should not be compared directly since the datasets are from different time spans and with different feature spaces. The initial benchmark primarily served to explore what hyper parameters should be used for the feature space exploration.

Table 4.11: **Summary of Benchmark: Algorithms Final**. Best scores for the different sweeps in Figure 4.4. This was done for the SMALL dataset.

Algorithm	Parameters	Accuracy
Liblinear	$C = 0.0133$	0.8154
LibSVM linear	$C = 0.0237$	0.8128
LibSVM RBF	$\gamma = 0.02, C = 4$	0.8248
Random Forest	$n_t = 95$	0.8124

4.3.2 Final classification

To give a final classification the optimal hyper parameters were used to benchmark the LARGE dataset. Since this dataset has imbalanced classes a cross-validation was set up with 10 runs. In each run, all exploited CVEs were selected together with an equal amount of randomly selected unexploited CVEs, to form an undersampled subset. The final result is shown in Table 4.12. In this benchmark, the models were trained on 80% of the data, and tested on the remaining 20%. Performance metrics are reported for both the training and test set. It is expected that the results are better for the training data, and some models like RBF and Random Forest get excellent results on classifying data already seen.

Table 4.12: **Benchmark: Algorithms Final.** Comparison of the different algorithms on the LARGE dataset split into training and testing data.

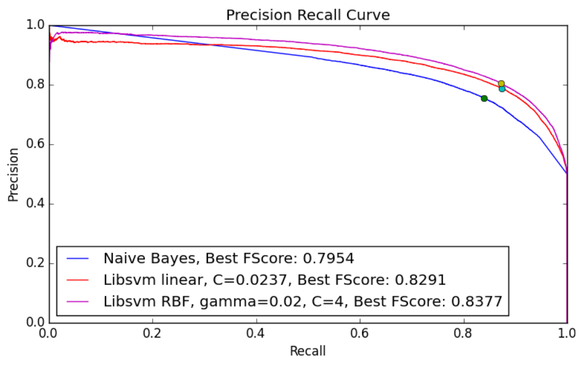
Algorithm	**Parameters**	**Dataset**	**Accuracy**	**Precision**	**Recall**	**F_1**	**Time**
Naive Bayes		train	0.8134	0.8009	0.8349	0.8175	0.02s
		test	0.7897	0.7759	0.8118	0.7934	
Liblinear	$C = 0.0133$	train	0.8723	0.8654	0.8822	0.8737	0.36s
		test	0.8237	0.8177	0.8309	0.8242	
LibSVM, linear	$C = 0.0237$	train	0.8528	0.8467	0.8621	0.8543	112.21s
		test	0.8222	0.8158	0.8302	0.8229	
LibSVM, RBF	$C = 4, \gamma = 0.02$	train	0.9676	0.9537	0.9830	0.9681	295.74s
		test	0.8327	0.8246	0.8431	0.8337	
Random Forest	$n_t = 80$	train	0.9996	0.9999	0.9993	0.9996	177.57s
		test	0.8175	0.8203	0.8109	0.8155	

4.3.3 Probabilistic classification

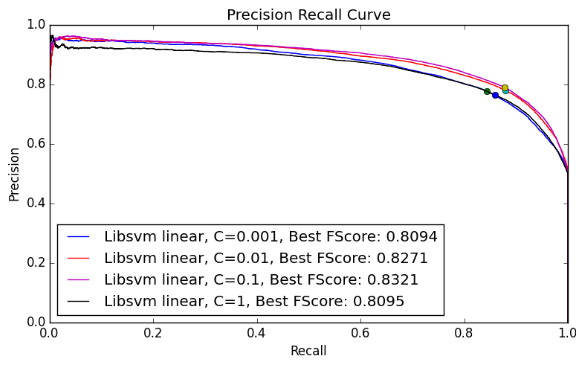
By using a probabilistic approach, it is possible to get class probabilities instead of a binary classification. The previous benchmarks in this thesis use a binary classification, and use a probability threshold of 50%. By changing this, it is possible to achieve better precision *or* recall. For example, by requiring the classifier to be 80% certain that there is an exploit, precision can increase, but recall will degrade. Precision recall curves were calculated and can be seen in Figure 4.5a. The optimal hyper parameters were found for the threshold 50%. This does not guarantee that they are optimal for any other threshold. If a specific recall or precision is needed for an application, the hyper parameters should be optimized with respect to that. The precision recall curve in Figure 4.5a for LibSVM is for a specific C value. In Figure 4.5b the precision recall curves are very similar for other values of C. The means that only small differences are to be expected.

We also compare the precision recall curves for the SMALL and LARGE datasets in Figure 4.6. We see that the characteristics are very similar, which means that the optimal hyper parameters found for the SMALL set also give good results on the LARGE set. In Table 4.13 the results are summarized and reported with log loss, best F_1-score, and the threshold where the maximum F_1-score is found. We see that the log loss is similar for the SVCs, but higher for Naive Bayes. The F_1-scores are also similar, but achieved at different thresholds for the Naive Bayes classifier.

Liblinear and Random Forest were not used for this part; Liblinear is not a probabilistic classifier, and the scikit-learn implementation *RandomForestClassifier* was unable to train a probabilistic version for these datasets.

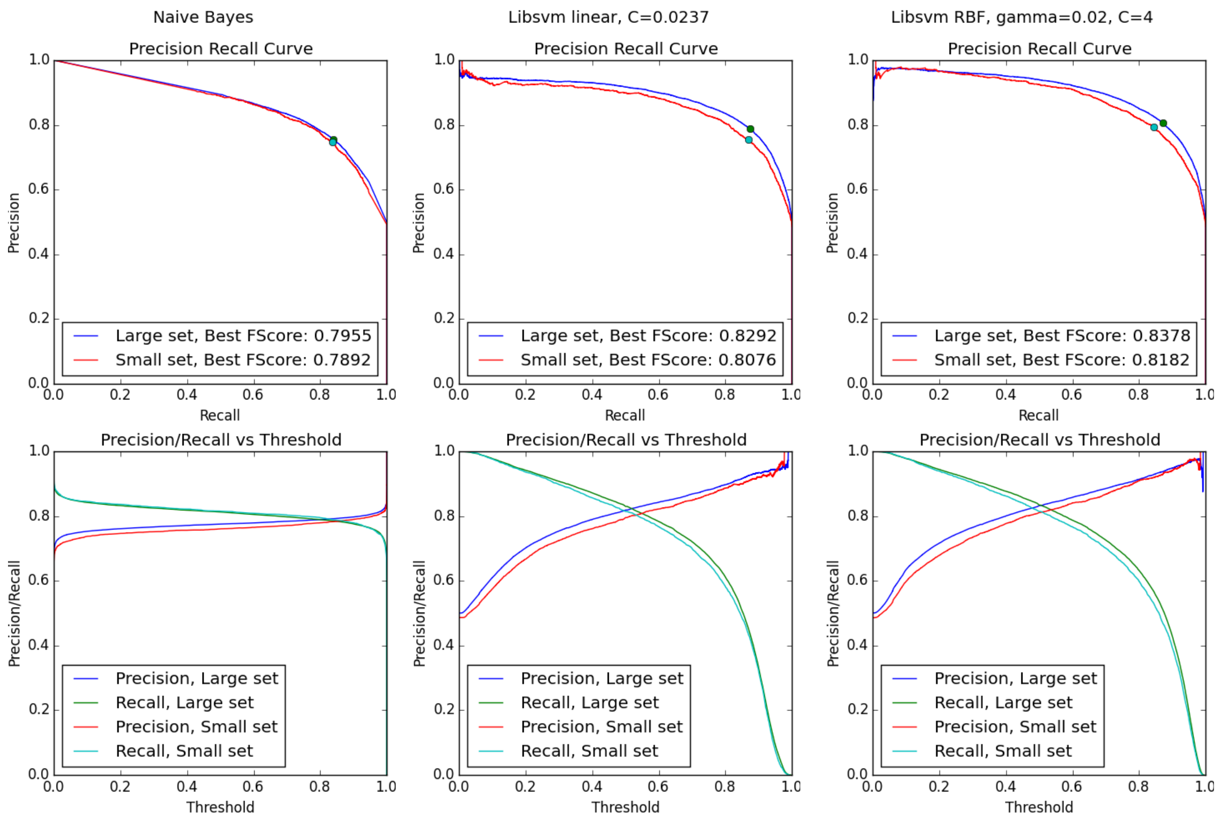


(a) Algorithm comparison.



(b) Comparing different penalties C for LibSVM linear.

Figure 4.5: **Precision recall curves** for the final round for some of the probabilistic algorithms. In Figure 4.5a SVCs perform better than the Naive Bayes classifier, with RBF being marginally better than the linear kernel. In Figure 4.5b curves are shown for different values of C.



(a) Naive Bayes (b) SVC with linear kernel (c) SVC with RBF kernel

Figure 4.6: **Precision recall curves** comparing the precision recall performance on the SMALL and LARGE dataset. The characteristics are very similar for the SVCs.

Table 4.13: **Summary of the probabilistic classification.** Best threshold is reported as the threshold where the maximum F_1-score is achieved.

Algorithm	**Parameters**	**Dataset**	**Log loss**	$\mathbf{F_1}$	**Best threshold**
Naive Bayes		LARGE	2.0119	0.7954	0.1239
		SMALL	1.9782	0.7888	0.2155
LibSVM, linear	$C = 0.0237$	LARGE	0.4101	0.8291	0.3940
		SMALL	0.4370	0.8072	0.3728
LibSVM, RBF	$C = 4, \gamma = 0.02$	LARGE	0.3836	0.8377	0.4146
		SMALL	0.4105	0.8180	0.4431

4.4 Dimensionality reduction

A dataset was constructed with $n_v = 20000$, $n_w = 10000$, $n_r = 1649$, and the base CWE and CVSS features, making $d = 31704$. The Random Projections algorithm in scikit-learn then reduced this to $d = 8840$ dimensions. For the Randomized PCA the number of components to keep was set to $n_c = 500$. Since this dataset has imbalanced classes a cross-validation was set up with 10 runs. In each run, all exploited CVEs were selected together with an equal amount of randomly selected unexploited CVEs, to form an undersampled subset.

Random Projections and Randomized PCA were then run on this dataset, together with a Liblinear ($C = 0.02$) classifier. The results, shown in Table 4.14, are somewhat at the same level as classifiers without any data reduction, but required much more calculations. Since the results are not better than using the standard datasets without any reduction, no more effort was put into investigating this.

Table 4.14: **Dimensionality reduction.** The time includes dimensionally reduction plus the time for classification. The performance metrics are reported as the mean over 10 iterations. All standard deviations are below 0.001.

Algorithm	**Accuracy**	**Precision**	**Recall**	$\mathbf{F_1}$	**Time** (s)
None	0.8275	0.8245	0.8357	0.8301	0.00+0.82
Rand. Proj.	0.8221	0.8201	0.8288	0.8244	146.09+10.65
Rand. PCA	0.8187	0.8200	0.8203	0.8201	380.29+1.29

4.5 Benchmark Recorded Future's data

By using Recorded Future's CyberExploit events as features for the vulnerabilities, it is possible to further increase the accuracy. Since Recorded Future's data, with a distribution in Figure 3.1, primarily spans over the last few years, the analysis cannot be done for the period 2005-2014. No parameters optimization was done for this set, since the main goal is to show that the results get slightly better if the RF data is used as features. An SVM Liblinear classifier was used with $C = 0.02$.

For this benchmark a feature set was chosen with CVSS and CWE parameters, $n_v = 6000$, $n_w = 10000$, and $n_r = 1649$. Two different date periods are explored; 2013-2014, and 2014 only. Since these datasets have imbalanced classes a cross-validation was set up with 10 runs like in Section 4.4. The test results can be seen in Table 4.15. The results get a little better if only data from 2014 are used. The scoring metrics become better when the extra CyberExploit events are used. However, this cannot be used in a real production system. When making an assessment on new vulnerabilities, this information will not be available. Generally, this also to some extent builds the answer into the feature matrix, since vulnerabilities with exploits are more interesting to tweet, blog, or write about.

Table 4.15: **Benchmark: Recorded Future's CyberExploits**. Precision, Recall, and F-score are for detecting the exploited label, and reported as the mean ± 1 standard deviation.

CyberExploits	**Year**	**Accuracy**	**Precision**	**Recall**	$\mathbf{F_1}$
No	2013-2014	0.8146 ± 0.0178	0.8080 ± 0.0287	0.8240 ± 0.0205	0.8154 ± 0.0159
Yes	2013-2014	0.8468 ± 0.0211	0.8423 ± 0.0288	0.8521 ± 0.0187	0.8469 ± 0.0190
No	2014	0.8117 ± 0.0257	0.8125 ± 0.0326	0.8094 ± 0.0406	0.8103 ± 0.0288
Yes	2014	0.8553 ± 0.0179	0.8544 ± 0.0245	0.8559 ± 0.0261	0.8548 ± 0.0185

4.6 Predicting exploit time frame

For the exploit time frame prediction the filtered datasets seen in Figures 3.5 and 3.6 were used; with respective dates between 2005-01-01 - 2014-12-31. For this round the feature matrix was limited to $n_v = 6000, n_w = 10000, n_r = 1649$, similar to the final binary classification benchmark in Section 4.3.2. CWE and CVSS base parameters were also used. The dataset was queried for vulnerabilities with exploit publish dates between 0 and 365 days after the CVE publish/public date. Four labels were used configured as seen in Table 4.16. For these benchmarks three different classifiers were used; Liblinear with $C = 0.01$; Naive Bayes; and a dummy classifier making a class weighted (stratified) guess. A classifier can be deemed useless if it behaves worse than random. Two different setups were used for the benchmarks to follow; subsampling and weighting.

For the first test, an equal amount of samples were picked from each class at random. The classifiers were then trained and cross-validated using stratified 5-Fold. The results are shown in Figure 4.7. The classifiers perform marginally better than the dummy classifier, and gives a poor overall performance. Without subsampling the whole dataset is used for classification, both with weighted and unweighted SVMs. The results are shown in Figure 4.8. The CERT data generally performs better for class 0-1, since it is very biased to that class. The other classes perform poorly. Naive Bayes fails completely for several of the classes, but will learn to recognize class 0-1 and 31-365 much better.

Table 4.16: Label observation counts in the different data sources used for the date prediction. Date difference from publish or public date to exploit date.

Date difference	Occurrences NVD	Occurrences CERT
0 - 1	583	1031
2 - 7	246	146
8 - 30	291	116
31 - 365	480	139

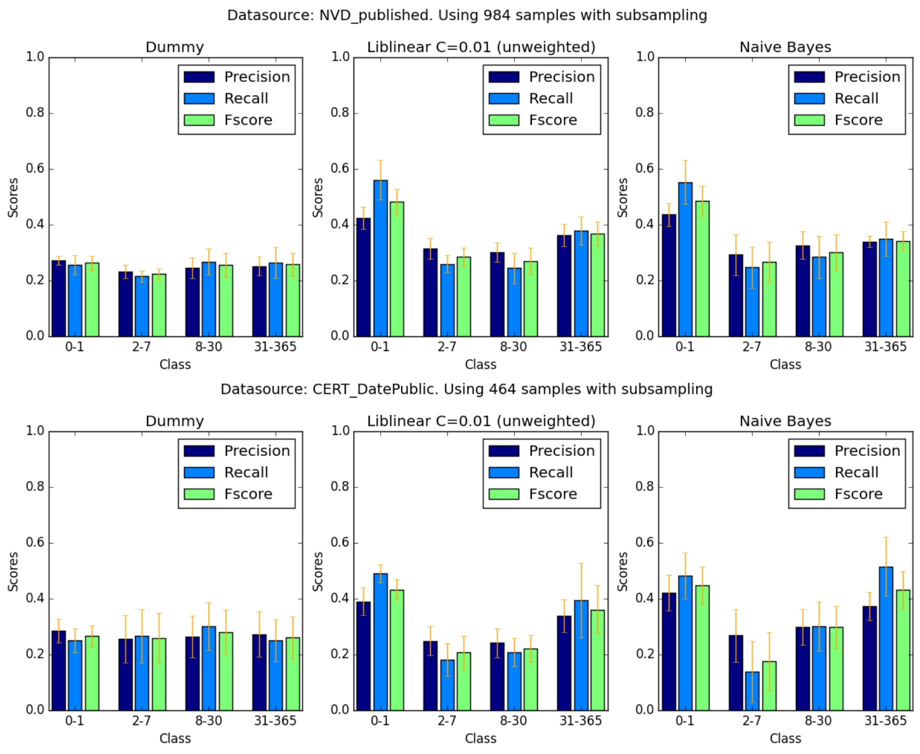


Figure 4.7: **Benchmark: Subsampling**. Precision, Recall, and F-score for the four different classes, with 1 standard deviation in the orange bars.

The general finding for this result section is that both the amount of data, and the data quality is insufficient in order to make any good predictions. In order to use date prediction in a production system, the predictive powers must be much more reliable. Different values for C were also tried briefly, but did not give any noteworthy differences. No more effort was put into investigating different techniques to improve these results.

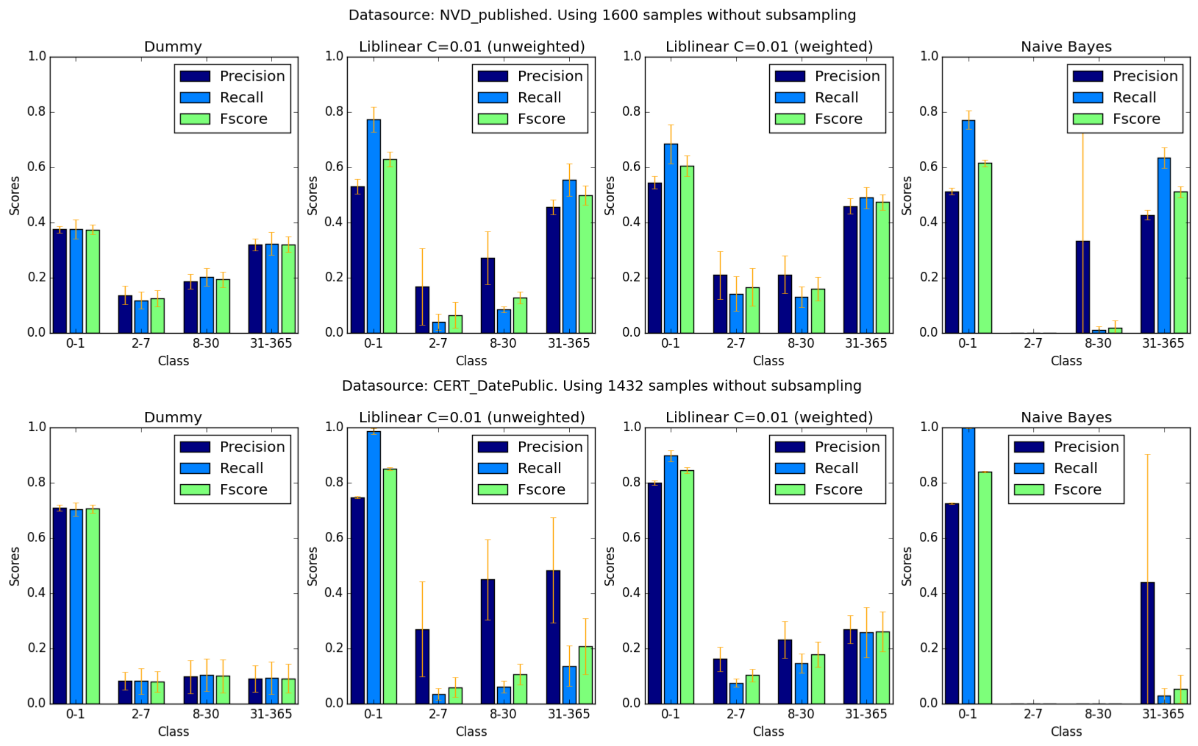


Figure 4.8: **Benchmark: No subsampling**. Precision, Recall, and F-score for the four different classes, with 1 standard deviation in the orange bars. The SVM algorithm is run with a weighted and an unweighted case.

5 Discussion & Conclusion

5.1 Discussion

As a result comparison, Bozorgi et al. (2010) achieved a performance of roughly 90% using Liblinear, but also used a considerable amount of more features. This result was both for binary classification and date prediction. In this work the authors used additional OSVDB data, which was not available in this thesis. The authors achieved a much better performance for time frame prediction, mostly due to the better data quality and amount of observations.

As discussed in Section 3.3, the quality of the labels is also doubtful. Is it really the case that the number of vulnerabilities with exploits is decreasing? Or are just fewer exploits reported to the EDB?

The use of open media data harvested by Recorded Future (CyberExploit events) shows potential for binary classifying older data, but will not be available for making predictions on new vulnerabilities. A key aspect to keep in mind is the information availability window. More information about vulnerabilities is often added to the NVD CVE entries as time goes by. To make a prediction about a completely new vulnerability the game will change, since not all information will be immediately available. The features that will be instantly available are base CVSS parameters, vendors, and words. It would also be possible to make a classification based on information about a fictive description, added vendors, and CVSS parameters.

References are often added later, when there is already information about the vulnerability being exploited or not. The important business case is to be able to make an early assessment based on what is known at an early stage, and update the prediction as more data is known.

5.2 Conclusion

The goal of this thesis was to use machine learning to examine correlations in the vulnerability data from the NVD and the EDB, and see if some vulnerability types are more likely to be exploited. The first part explains how binary classifiers can be built to predict exploits for vulnerabilities. Using several different ML algorithms, it is possible to get a prediction accuracy of 83% for the binary classification. This shows that to use the NVD alone is not ideal for this kind of benchmark. Predictions on data were made using ML algorithms such as SVMs, kNN, Naive Bayes, and Random Forests. The relative performance of several of those classifiers is marginal with respect to metrics such as accuracy, precision, and recall. The best classifier with respect to both performance metrics and execution time is SVM Liblinear.

This work shows that the most important features are common words, references, and vendors. CVSS parameters, especially CVSS scores, and CWE-numbers are redundant when a large number of common words are used. The information in the CVSS parameters and CWE-number is often contained within the vulnerability description. CAPEC-numbers were also tried, and yielded similar performance as the CWE-numbers.

In the second part, the same data was used to predict an exploit time frame as a multi-label classification problem. To predict an exploit time frame for vulnerabilities proved hard using only unreliable data from the NVD and the EDB. The analysis showed that using only public or publish dates of CVEs or EDB exploits were not enough, and the amount of training data was limited.

5.3 Future work

Several others (Allodi and Massacci, 2013, Frei et al., 2009, Zhang et al., 2011) also concluded that the NVD is of poor quality for statistical analysis (and machine learning). To be able to improve research in this area, we recommend the construction of a modern open source vulnerability database, where security experts can collaborate, report exploits, and update vulnerability information easily. We further recommend that old data should be cleaned up and dates should be verified.

From a business perspective it could be interesting to build an adaptive classifier, or estimator, for assessing new vulnerabilities based on what data that can be made available. For example, to design a web form where users can input data for a vulnerability and see how the prediction changes based on the current information. This could be used in a production system for evaluating new vulnerabilities. For this to work, we would also need to see how the classifiers trained in this thesis would perform on new and information-sparse observations and see how predictions change when new evidence is presented. As has been mentioned previously, the amount of available information for a vulnerability can be limited when the CVE is created, and information is often updated later.

Different commercial actors such as Symantec (The WINE database) and the OSVDB have data that could be used for more extensive machine learning analysis. It is also possible to used more sophisticated NLP algorithms to extract more distinct features. We are currently using a bag-of-words model, but it would be possible to use the semantics and word relations to get better knowledge of what is being said in the summaries.

The author is unaware of any studies on predicting which vulnerabilities and exploits that will be used in exploit kits in the wild. As shown by Allodi and Massacci (2012), this is only a small subset of all vulnerabilities, but are those vulnerabilities that are going to be most important to foresee.

Bibliography

Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, June 2003.

Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In *Machine Learning: ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 39–50. Springer Berlin Heidelberg, 2004.

Luca Allodi and Fabio Massacci. A preliminary analysis of vulnerability scores for attacks in wild: The ekits and sym datasets. In *Proceedings of the 2012 ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '12, pages 17–24, New York, NY, USA, 2012. ACM.

Luca Allodi and Fabio Massacci. Analysis of exploits in the wild. or: Do cybersecurity standards make sense?, 2013. Poster at IEEE Symposium on Security & Privacy.

Luca Allodi and Fabio Massacci. The work-averse attacker model. In *Proceedings of the 2015 European Conference on Information Systems (ECIS 2015)*, 2015.

D. Barber. *Bayesian Reasoning and Machine Learning.* Cambridge University Press, 2012.

Samy Bengio, Johnny Mariéthoz, and Mikaela Keller. The expected performance curve. In *International Conference on Machine Learning, ICML, Workshop on ROC Analysis in Machine Learning*, 0 2005.

Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 105–114, New York, NY, USA, 2010. ACM.

Joshua Cannell. Tools of the trade: Exploit kits, 2013. URL `https://blog.malwarebytes.org/intelligence/2013/02/tools-of-the-trade-exploit-kits/`. Last visited: 2015-03-20.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, sep 1995.

Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

Tudor Dumitras and Darren Shou. Toward a standard benchmark for computer security research: The worldwide intelligence network environment (wine). In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '11, pages 89–96, New York, NY, USA, 2011. ACM.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. Large-scale vulnerability analysis. In *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, LSAD '06, pages 131–138, New York, NY, USA, 2006. ACM.

Stefan Frei, Dominik Schatzmann, Bernhard Plattner, and Brian Trammell. Modelling the security ecosystem- the dynamics of (in)security. In *WEIS*, 2009.

Vaishali Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2, 2012.

Rick Gordon. Note to vendors: Cisos don't want your analytical tools, 2015. URL `http://www.darkreading.com/vulnerabilities---threats/note-to-vendors-cisos-dont-want-your-analytical-tools/a/d-id/1320185`. Last visited: 2015-04-29.

William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 287–296. ACM, 2006.

Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47 – 68, 2011.

Fabio Massacci and Viet Hung Nguyen. Which is the right source for vulnerability studies?: An empirical analysis on mozilla firefox. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, MetriSec '10, pages 4:1–4:8, New York, NY, USA, 2010. ACM.

Peter Mell, Karen Scarfone, and Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0.* NIST and Carnegie Mellon University, 2007. URL `http://www.first.org/cvss/cvss-guide.html`. Last visited: 2015-05-01.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

Paul Oliveria. Patch tuesday - exploit wednesday, 2005. URL `http://blog.trendmicro.com/trendlabs-security-intelligence/patch-tuesday-exploit-wednesday/`. Last visited: 2015-03-20.

Andy Ozment. Improving vulnerability discovery models. In *Proceedings of the 2007 ACM Workshop on Quality of Protection*, QoP '07, pages 6–11, New York, NY, USA, 2007. ACM.

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2 (6):559–572, 1901.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

W. Richert. *Building Machine Learning Systems with Python.* Packt Publishing, 2013.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05. URL `citeseer.ist.psu.edu/sahami98bayesian.html`.

Woohyun Shim, Luca Allodi, and Fabio Massacci. Crime pays if you are just an average hacker. pages 62–68. IEEE Computer Society, 2012.

Jonathon Shlens. A tutorial on principal component analysis. *CoRR*, 2014.

Mark Stockley. How one man could have deleted every photo on facebook, 2015. URL `https://nakedsecurity.sophos.com/2015/02/12/`

`how-one-man-could-have-deleted-every-photo-on-facebook/`. Last visited: 2015-03-20.

David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.

Su Zhang, Doina Caragea, and Xinming Ou. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *Proceedings of the 22nd International Conference on Database and Expert Systems Applications - Volume Part I*, DEXA'11, pages 217–231, Berlin, Heidelberg, 2011. Springer-Verlag.