FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Unsupersived Instance Selection for Malware Detection
Student:	Mehmet Efe Zorlutuna
Supervisor:	Mgr. Martin Jureček, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security 2021
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

Machine learning algorithms are widely used in the area of malware detection. With the growth of sample amounts, training of classification algorithms becomes more and more expensive. The problem to be solved is selecting representative unlabeled samples from large training data sets without reducing the accuracy. This work aims to solve this problem in an unsupervised learning fashion.

Instructions:
1) Study the state-of-the-art unsupervised instance selection algorithms.
2) Try to propose new or modify existing unsupervised instance selection algorithms.
3) Use existing libraries or implement at least two unsupervised instance selection algorithms for malware detection.
4) Compare and discuss the experimental results in terms of the reduction rate, the accuracy, and the computational time.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 9 February 2023 in Prague.

Master's thesis

UNSUPERSIVED INSTANCE SELECTION FOR MALWARE DETECTION

BS Mehmet Efe Zorlutuna

Faculty of Information Technology
Department of Information Security
Supervisor: Mgr. Martin Jureček Ph.D.
May 4, 2023

Contents

List of Figures

List of Tables

List of Algorithms

I thank my supervisor for introducing me to instance selection algorithms and malware detection.
I am grateful to my family for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Praze on May 4, 2023 .....

Abstract

This work proposes a new unsupervised instance selection algorithm for malware detection and evaluates its effectiveness. The proposed algorithm selects informative instances from unlabelled data to train a machine learning model, which is expected to improve its accuracy and efficiency. Experiments show a performance comparison of the algorithm and an existing unsupervised instance selection algorithm.

Keywords Malware Detection, Machine Learning, Unsupervised Instance Selection

Abstrakt

Tato práce navrhuje nový algoritmus výběru instance bez dozoru pro detekci malwaru a vyhodnocuje jeho efektivitu. Navržený algoritmus zvolí informativní výběr z neoznačených dat, aby natrénoval model strojového učení, od kterého se očekává zlepšení jeho přesnosti a účinnosti. Experimenty ukazují srovnání výkonu algoritmu a existujícího algoritmu výběru instance bez dozoru.

Klíčová slova Detekce Malwaru, Strojové Učení, Výběr Instance Bez Dozoru

Introduction

Artificial intelligence and machine learning are becoming increasingly popular and are being used in various areas, including malware detection. However, the performance of machine learning algorithms is highly dependent on the quality of the training dataset. Creating a compact and highly descriptive training dataset that yields high performance with machine learning algorithms can be challenging. Instance selection algorithms can be used to create a more manageable and accurate training dataset. However, the majority of state-of-the-art instance selection algorithms require labeled data, which can be expensive and time-consuming to obtain.

This work aims to propose an unsupervised instance selection algorithm and compare it with another unsupervised instance selection algorithm.

Chapter 1 provides an overview of the key concepts and technologies relevant to this study, including malware detection, machine learning, and instance selection algorithms.

Chapter 2 presents our proposed instance selection algorithm and provides a description of each step of the algorithm, including how it selects a representative subset of the original dataset.

Chapter 3 provides implementation notes that help readers understand the technical details of how our proposed algorithm is implemented.

Chapter 4 describes the methods we used to evaluate the performance of our proposed algorithm, including the evaluation metrics and the comparison to another unlabeled instance selection algorithm.

Chapter 5 proposes implementation notes for setting up the tests..

Chapter 6 presents the results of our experiments, including a detailed analysis of the accuracy, runtime, and reduction rate of both the proposed algorithm and the compared algorithm, offering insights into the strengths and weaknesses of each algorithm.

In the final chapter, we compare the proposed instance selection algorithm to the selected unlabeled instance selection algorithm and discuss the results. We provide insights into the strengths and weaknesses of our proposed algorithm and suggest areas for further research.

Motivation

Machine learning algorithms are essential in the field of malware detection, as they rely on training data sets during the training phase. The accuracy and size of the training data set are critical factors that impact the performance and results of the algorithm. A training data set's accuracy is determined by how well it represents the patterns of similarly classified instances and the differences between differently classified instances.

Additionally, the size of the training data set is also important because it affects the time taken by the machine learning algorithm to yield results. With the increasing number of malware instances detected every day, the algorithm's yielding time becomes crucial. If the yielding time is too long, it can significantly decrease the accuracy of the machine-learning algorithm. Therefore, it is important to have a training data set that is accurately descriptive without being excessively large.

One way to address the issue of the size of the training data set is by using instance selection algorithms. These algorithms reduce the size of the training data set during the preprocessing step of machine learning. The goal of instance selection is to reduce the original dataset to a manageable size, which improves the performance of the machine learning algorithm while maintaining the accuracy of the algorithm.

Labeling the data is another challenge in the field of malware detection. However, a proposed algorithm inspired by Parallel Instance Filtering (PIF) [1] addresses this issue by using clustering instead of labels to determine the enemies. This algorithm does not require labeled data and is thus a valuable approach to addressing the labeling problem.

The PIF algorithm reduces the size of the training data set for malware detection by using labeled data to determine the enemies for each instance and eliminate redundant data. In contrast, the proposed algorithm clusters instances based on their features to determine the enemies with respect to cluster and eliminate redundant data.

To sum up, machine-learning algorithms are important for malware detection, but the accuracy and size of training data sets are critical. The training data set's accuracy and size affect the algorithm's performance and yielding time, which is crucial as new malware is detected daily. Instance selection algorithms reduce the training data set's size while maintaining accuracy, and the proposed algorithm inspired by PIF addresses the labeling problem by using clustering to determine enemies without labeled data.

Chapter 1
Background Information

1.1 Artificial intelligence

1.1.1 Definition of Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science that deals with the creation of intelligent machines that can perform tasks that usually require human-like intelligence, such as perception, reasoning, learning, and problem-solving. AI involves developing algorithms and models that can make predictions, take actions, and adapt to new situations based on data and feedback.

Intelligence is a complex concept that has different definitions depending on the context. In general, intelligence can be described as the ability to acquire and apply knowledge and skills to solve problems and achieve goals. In the case of machines, intelligence can refer to various capabilities, such as pattern recognition, natural language processing, decision-making, and creativity.

AI has numerous applications in various industries, including healthcare, finance, transportation, and entertainment. Some of the most popular implementations of AI are in image recognition, speech recognition, playing games, driving cars, text generation, image generation, and malware detection.

Image recognition, for example, involves using computer algorithms to identify and classify objects within an image. Speech recognition involves converting spoken words into text or commands that a computer can understand. Playing games involves developing AI algorithms that can learn and improve their performance over time by playing against themselves or human players. Driving cars involves creating autonomous vehicles that can perceive their environment and make decisions based on the data they receive. Text generation involves generating human-like text based on a given prompt or topic, while image generation involves creating new images based on input data.

AI algorithms can be trained to detect malware by analyzing patterns in the code and behavior of the software. This involves creating models that can distinguish between benign and malicious software based on features such as file size, file type, system calls, network activity, and code structure. Machine learning techniques, such as deep learning and natural language processing, can be used to automatically learn these patterns and improve the accuracy of malware detection.

One of the key features of AI is its ability to operate autonomously, without human intervention. AI systems can learn from data and feedback, make decisions, and perform tasks without requiring explicit instructions or guidance from humans. However, human intervention is still necessary in the early stages of development, where programmers need to specify the task and provide the necessary data for the machine to learn from.

1.1.2 Weak AI and Strong AI

Currently, all existing AI is considered to be weak AI. Weak AI, also known as narrow AI, refers to AI that is designed to perform specific tasks in a limited area. This type of AI is highly specialized and can excel in performing tasks such as speech recognition, image classification, game playing, and mawlware detection. Weak AI systems are programmed to follow specific rules and algorithms, and they do not have the ability to think or reason beyond their programmed tasks.

On the other hand, strong AI, also known as artificial general intelligence (AGI), is a hypothetical form of AI that is capable of reasoning, planning, self-consciousness, and problem-solving at the human level. Strong AI would be able to understand and learn from its environment, think creatively, and perform any intellectual task that a human can do. This type of AI would be able to adapt to new situations and solve problems beyond its initial programming.

Currently, strong AI is a theoretical concept, and no system has been developed that can perform at this level. The development of strong AI is a challenging task that requires the integration of various AI disciplines, such as machine learning, natural language processing, computer vision, and robotics, among others. Moreover, there are several ethical concerns regarding the development of strong AI, including the possibility of superintelligence that could pose a threat to humanity.

1.2 Machine Learning

1.2.1 Definition of Machine Learning

Machine learning (ML) is a subfield of artificial intelligence that focuses on developing algorithms and models that enable computers to learn and improve their performance based on experience. In machine learning, computers are trained on large datasets and are able to learn patterns and relationships within the data, which they can then use to make predictions or decisions.

Machine learning has a wide range of applications, including image recognition, natural language processing, fraud detection, recommendation systems, and autonomous vehicles, malware detection among others. Machine learning algorithms are used to train models that can accurately recognize objects within images, understand and generate natural language, and make decisions based on complex data.

As the amount of data generated by individuals and organizations continues to grow, machine learning is becoming increasingly important for making sense of this data and extracting useful insights. The development of more advanced machine learning algorithms and techniques is expected to drive further innovation and advancements in AI.

1.2.2 Types of Machine Learning

There are four different types of machine learning, supervised learning, unsupervised learning, semi-supervised learning, and reinforced learning.

1.2.2.1 Supervised Machine Learning

In supervised learning during the learning phase, the machine learning algorithm uses a training data set with classified instances, and the algorithm constructs the link between the pattern and classification. Some popular techniques used in supervised learning are support-vector machines, Naive Bayes, linear discriminant analysis, decision trees, k-nearest neighbors algorithm, neural networks, and similarity learning.

1.2.2.2 Unsupervised Machine Learning

Unsupervised learning does not use classifications, it uses unclassified data to analyze and cluster or disclose the association of instances by patterns in the unlabeled training data set. Some popular techniques used in unsupervised learning are k-means clustering, k-nearest neighbors algorithm, hierarchal clustering, anomaly detection, and neural networks.

1.2.2.3 Semi-supervised Machine Learning

Semi-supervised learning uses a combination of supervised and unsupervised approaches, only a limited chunk of the training data set is classified, and the algorithm explores links between the classified and unclassified data. Some methods applied in semi-supervised learning are Generative models low-density separation, and Laplacian regularization.

1.2.2.4 Reinforcement Machine Learning

Reinforcement machine learning is a training method performed by rewarding the desired behavior and punishing the undesired behavior, for instance, in a chess game picking a queen is more desirable than picking a pawn, thus corresponding reward of picking a queen is more than picking a pawn for a reinforcement learning algorithm. Some reinforcement learning algorithms are direct utility estimation, adaptive dynamic programming, temporal-difference learning, and exploration.

1.3 Instance Selection

1.3.1 Definition of Instance Selection

Instance selection is a technique in machine learning that involves selecting a subset of instances from a dataset to use as training data for a machine learning model. The goal of instance selection is to improve the performance of a machine learning algorithm by reducing the size of the training set and removing irrelevant or redundant instances. This can lead to faster training times, improved model accuracy, and more efficient use of computing resources.

Instance selection can be useful in a variety of settings. For example, in situations where the dataset is very large, instance selection can be used to reduce the size of the dataset, making it more manageable for a machine learning algorithm to process. In addition, instance selection can be used to improve the performance of a machine learning algorithm when the dataset contains a large number of irrelevant or redundant instances, which can negatively impact the accuracy of the model.

1.3.2 Advantage of Instance Selection

There are several reasons why instance selection might be useful in machine learning. For example, if a training dataset contains a large number of instances that are irrelevant to the task at hand, using all of them as training data can result in poor model performance and longer training times. By selecting a subset of instances that are most relevant to the task, instance selection can help to improve model accuracy and reduce training time.

Overall, instance selection can be a useful technique for improving the performance of machine learning models, particularly in situations where the original dataset is large or noisy.

1.3.3 Methods of Instance Selection

There are different methods for instance selection, but they generally involve choosing a subset of the training data that captures the important patterns and relationships in the data, while discarding the less informative instances. This can be achieved through various criteria such as diversity, density, or relevance. Instance selection techniques can be divided into two categories supervised and unsupervised. Additionally, they can be divided into two categories as filtering and wrapper methods.

1.3.4 Supervised and Unsupervised Instance Selection

Supervised instance selection methods use class labels to guide the selection process. The goal of these methods is to select instances that are most informative for the learning algorithm in terms of the classification task. The intuition behind supervised instance selection is that instances that are close to the decision boundary of a classifier are more informative than those that are far away. This is because the decision boundary is the area where the classifier is uncertain about the correct class label, and instances in this region can help the classifier to make better predictions.

There are several supervised instance selection methods. Instance-based learning is a popular method that involves selecting instances that are close to the decision boundary of a classifier. This method is based on the assumption that instances that are close to the boundary are more informative than those that are far away. Selective sampling is another method that involves selecting instances that are difficult to classify. This method is based on the intuition that difficult instances are more informative than easy ones, as they are likely to contain information that can help the classifier to make better predictions.

Unsupervised instance selection methods, on the other hand, do not rely on class labels to guide the selection process. Instead, they use distance or similarity measures to identify relevant instances. The goal of unsupervised instance selection is to group similar instances together and remove redundant or irrelevant instances. This can help to reduce the size of the dataset and improve the performance of the learning algorithm.

There are several unsupervised instance selection methods. K-means clustering is a popular method that involves grouping similar instances together based on their distance to a set of centroids. This method is based on the assumption that instances that are close together are more similar than those that are far apart. Density-based clustering is another method that involves selecting instances that are located in high-density regions of the data. This method is based on the intuition that instances that are located in high-density regions are more representative of the data than those that are located in low-density regions.

1.3.5 Filtering and Wrapper Methods of Instance Selection

Filtering methods, for instance, selection involve selecting a subset of instances from the original dataset based on certain criteria, without using a specific learning algorithm. These methods typically involve analyzing the statistical properties of the data or the performance of the learning algorithm on the dataset. For example, a filtering method might involve removing instances that are missing values or have high levels of noise.

Filtering methods are typically faster and less computationally expensive than wrapper methods, as they do not require running a specific learning algorithm. However, they may not always lead to the best performance, as they do not take into account the specific characteristics of the learning algorithm being used.

Wrapper methods for instance selection, on the other hand, involve using a specific learning algorithm to evaluate the quality of a subset of instances. These methods typically involve training a learning algorithm on different subsets of instances and selecting the subset that leads

to the best performance. Wrapper methods can be more computationally expensive than filtering methods, as they require training the learning algorithm multiple times.

Overall, both filtering and wrapper methods, for instance, selection have their advantages and disadvantages, and the choice between them depends on the specific problem being addressed and the resources available for computing. The goal of instance selection is to reduce the size of the dataset without significantly affecting the performance of the learning algorithm, which can lead to faster training times, improved model accuracy, and more efficient use of computing resources.

1.3.6 State-of-Art Instance Selection Algorithms

Some examples of state-of-the-art instance selection algorithms include Condensed Nearest Neighbor (CNN)[2], Edited Nearest Neighbor ENN)[3], Iterative Case Filtering algorithm (ICF)[4], and Discriminative Random Over-Sampling Projection (DROP)[5].

- CNN is an instance selection algorithm that starts with a small subset of instances and iteratively adds new instances that are misclassified by the current model until no more instances can be added.
- ENN is an instance selection algorithm that removes instances that are misclassified by their nearest neighbors.
- ICF algorithm is a method for feature selection in machine learning that iteratively removes the least informative features until a stopping criterion is met.
- DROP generates a reduced dataset with higher discriminatory power to improve learning algorithm performance while reducing the computational cost.

1.3.7 Nimble Instance Selection Algorithm

A more recent instance selection algorithm that has been compared to the proposed NCE algorithm is NIS, which stands for Nimble Instance Selection. The NIS algorithm uses hyper-rectangles to discard instances, but instead of finding hyper-rectangles directly, it leverages the concept of data scaling in statistics to indirectly form hyper-rectangles. This approach avoids the computational cost of calculating the positions of real hyper-rectangles, which can slow down the instance selection process. By exploiting this concept, the algorithm can efficiently and effectively select representative instances without sacrificing performance.

1.3.8 Deeper Definition of Unsupervised Learning

Unsupervised learning algorithms are a class of machine learning techniques that aim to find patterns or relationships in data without the use of labeled examples. They have a number of advantages over other machine learning techniques, including the fact that they depend less on human intervention and can perform their analysis using completely unlabeled data.

1.3.8.1 Advantages of Unsupervised Learning

This can be particularly useful in cases where labeled data is scarce or expensive to obtain. One of the main advantages of unsupervised learning algorithms is that they can discover hidden structures and patterns in the data that may not be apparent to humans. For example, clustering algorithms can group similar data points together, even if they do not have a pre-defined label or category. This can be useful in a wide range of applications, from identifying customer segments in marketing to detecting anomalous behavior in cyber security.

1.3.8.2 Challanges of Unsupervised Learning

However, while unsupervised learning algorithms have many benefits, they also come with their own set of challenges. One major challenge is the potential for computational complexity and resource consumption due to the high volume of training data. This can lead to longer execution times and higher resource consumption, which can be particularly challenging when working with large datasets.

Additionally, due to the uncertainty of the training data, unsupervised learning algorithms may produce inaccurate results more frequently than supervised learning algorithms. This is because there is no ground truth against which the algorithm can be evaluated. As a result, it is important to carefully evaluate the results of unsupervised learning algorithms and to use additional methods to verify their accuracy.

Furthermore, while unsupervised learning algorithms can make analyses with unlabeled datasets, the evaluation of results may still require labels from the training set. This is because, without labels, it may be difficult to assess the accuracy and validity of the results. In some cases, labels may need to be added manually, which can be time-consuming and expensive.

Another issue with unsupervised learning algorithms is transparency. Even if the quality of clusters produced by the algorithm is high, the ground truth information about the instances may be unclear. For example, a cluster with high quality may represent a malware type, but further analysis is required to identify the specific type of malware. This can be particularly challenging in cases where the data is sensitive or where the results of the analysis could have significant real-world implications.

1.3.8.3 Sum-Up

While unsupervised learning algorithms have numerous benefits, they also present unique challenges. Researchers must carefully consider these challenges when deciding which machine-learning techniques to use for their particular applications. It is also important to continue developing new and improved algorithms that can overcome these challenges and to carefully evaluate the accuracy and validity of the results produced by unsupervised learning techniques.

1.4 Malware

1.4.1 Definition of Malware

Malware, which is short for malicious software, is a type of software that has been created by cyber threat actors to perform activities without the owner's consent. Malware can take many forms, including viruses, Trojans, worms, spyware, adware, and ransomware. These activities can range from simply stealing information to taking full control of a victim's device or network. Malware can be used for a variety of purposes, including stealing sensitive information, disrupting computer systems, and causing financial damage.

There are many types of cyber threat actors who create and distribute malware. These include nation-states, cybercriminals, hacktivists, terrorist groups, thrill-seekers, and even insiders. Nation-states may create and use malware for espionage purposes, while cybercriminals may use malware for financial gain by stealing sensitive information or holding it for ransom. Hacktivists may use malware to disrupt the operations of a targeted organization or to promote a particular cause. Terrorist groups may use malware as part of their efforts to carry out attacks, while thrill-seekers may create and distribute malware simply for the challenge of doing so. Insiders, such as disgruntled employees, may use malware to carry out acts of sabotage against their employers.

There are many techniques that cyber threat actors use to distribute malware, including phishing emails, malicious websites, drive-by downloads, and infected software downloads. Once

installed on a device or network, malware can be difficult to detect and remove, as it often has the ability to evade detection by antivirus software and other security measures.

1.4.2 Motivation of Malware

Malware can be created and used for a variety of reasons depending on the actor's intentions ranging from financial gain to political motivations.

1.4.2.1 Intrusion

One of the primary uses of malware is an intrusion. This involves gaining unauthorized access to a victim's device or network and collecting sensitive information about them. Cybercriminals may use this information to commit identity theft, financial fraud, or other types of cybercrime. Nation-states may use this information for espionage purposes, to gain a strategic advantage in global politics or business.

1.4.2.2 Disruption

Another use of malware is disruption. This involves making the victim's device or network unusable, usually through the use of ransomware or denial-of-service attacks. Ransomware is a type of malware that encrypts the victim's data and demands a ransom payment in exchange for the decryption key. Denial-of-service attacks involve flooding the victim's network with traffic, effectively rendering it unusable.

1.4.2.3 Financial Gain

Financial gain is another common motivation for cyber threat actors. Malware can be used to steal intellectual property, financial data, or other valuable information. This information can then be sold on the dark web to other cybercriminals or nation-states for profit. Cybercriminals may also use malware to conduct phishing scams, which involve tricking victims into giving away their financial or personal information.

1.4.2.4 Vandalism

Vandalism is another use of malware. This involves the destruction of a victim's device or network, either for personal reasons or as part of a larger cyber attack. This type of attack can be particularly devastating for small businesses or individuals who rely on their devices for work or personal use.

1.4.2.5 Steal Hardware Resources

Finally, some cyber threat actors may use malware to steal hardware resources. This involves using a victim's device to mine cryptocurrencies or perform other resource-intensive tasks. This can result in slow performance and increased energy consumption, as well as potential damage to the device itself.

1.5 Malware Detection

1.5.1 Definition of Malware Detection

Malware detection is a crucial part of cybersecurity, as it helps to identify and remove malicious software from devices before it can cause harm. The process of malware detection involves

analyzing the behavior of software to determine whether it is behaving in a way that is consistent with known malware. This can be done using a variety of techniques, including signature-based detection, behavior-based detection, and machine learning-based detection.

1.5.2 Challenges of Malware Detection

1.5.2.1 What is Malware?

The issue of malware detection can also give rise to deeper discussions about ethics and consent. For example, consider the act of mining cryptocurrency. While this activity is not inherently malicious, using a device without the owner's acknowledgment or consent is a clear violation of their privacy and property rights. If the victim has not given consent to the use of their device for cryptocurrency mining, then the act can be considered malicious. However, if the victim has given their consent without fully understanding the implications of their actions (for instance, by accepting the terms and conditions of an app without reading them), then the operation may be undesirable but not necessarily a crime.

1.5.2.2 False Positives

Another challenge of malware detection is the risk of false positives. Malware detection software relies on various heuristics and algorithms to identify malicious software. However, these algorithms can sometimes classify benign software as malicious, which can lead to false accusations and even lawsuits. For instance, antivirus software might flag a legitimate system file as a virus, leading to the file being deleted and causing the system to malfunction. In some cases, this can result in serious consequences, such as lost data or system downtime.

1.5.3 Solution for Challenges

To mitigate these challenges, it is essential to develop reliable and accurate malware detection systems. This involves using a range of techniques such as signature-based detection, behavioral analysis, and machine learning. Signature-based detection involves matching the code of a file or program against a database of known malware signatures. The behavioral analysis involves monitoring the behavior of a program to detect suspicious activity, such as attempts to modify system files or connect to malicious servers. Machine learning involves training a model to recognize patterns of behavior that are indicative of malware.

1.5.3.1 User Awareness

In addition to developing effective malware detection systems, it is also important to ensure that users are informed about the risks and implications of malware detection. This includes providing clear and concise information about the types of malware that may be detected, the consequences of false positives, and the importance of obtaining consent before performing any operations on a user's device.

1.5.4 Sum-Up

To conclude, malware detection is a critical defensive measure used to protect against a wide range of threats. However, the issue of malware detection can also give rise to deeper discussions about ethics and consent. To develop reliable and accurate malware detection systems, it is essential to use a range of techniques such as signature-based detection, behavioral analysis, and machine learning.

Additionally, it is important to ensure that users are informed about the risks and implications of malware detection and that their consent is obtained before performing any operations on their device.

1.6 Signature-Based Malware Detection

Signature-based malware detection works depending on the signature of the software. Those signatures are created using a hashing algorithm, for instance, Secure Hash Algorithms (SHA) as widely used. These signatures are kept in big databases for later use after creation. On encounter of new software on a computer, signature-based detection software recreates the signature of the new software and checks the signature in the database. However, this approach is not perfect, and the hassle of maintenance of this technique becomes more difficult over time.

1.6.1 Challenges of Malware Detection

1.6.1.1 Database Ownership

For new implementations, if there is no database created, owned, or usable by the implementation, the implementation depends on third-party databases. Using third-party databases may create conflicts because a signature might have a different classification in different databases, because the definition of malware changes by the policy of the company or service provider, for instance, advertisement software may be undesirable from one point of view but not harmful to the computer, but they are still potentially unwanted applications.

1.6.1.2 Maintenence of Malware Database

Beyond that, the maintenance of an already going project is a growing problem. Every day more than at least 450 thousand malware and potentially unwanted applications are created and detected [6]. So, maintenance of the database is one of the problems since old signatures should be kept, so every day these databases grow bigger, and every day the requirement for more maintenance of data increases. Besides maintenance, an excessive amount of data is costly to store and difficult to migrate, which causes restrictions not only on the information technology side but also on the business side. Migration of data means the process of transferring the data from one database to another one.

1.6.1.3 New Malware Signatures

Beyond maintenance, newly introduced software is useless for signature-based malware detection. To be able to detect malware by signature, its signature should be stored, and a definition of software corresponding to that signature in the database. Therefore, the data should be classified before using signature-based malware detection. Moreover, the working principle of malware is important information to classify them, for further analysis. There are mainly five different malware families. Most of the newly introduced malware is a descendant of old malware.

- Virlock is a family of Ransomware which means a malware type that typically encrypts the main storage of the victim's machine and requests ransom for decryption.
- Virut is a family of viruses that creates botnets. A computer virus is a type of malware that requires another software to be executed. Botnets are internet-connected network devices that are orchestrated by attackers, typically used for DoS attacks.
- Allaple is a family of polymorphic network worms and performs Denial-of-Service (DoS) attacks. Worms are software that does not depend on other software to run them and

have the ability to clone themselves in another node of networks. Polymorphic means, the software changes itself on each generation, which makes it harder to detect. DoS attacks characteristically block legitimate usage of service on target.

- Dinwod is a family of malware that secretly downloads and installs other malware. They are trojan horses which are malware that deceives users as it is benign software.

- Vundo family refers to either trojan horses or worms to display pop-up advertisements on the victim's screen.

1.6.1.4 Obfuscation of Malware

Besides the working principle, malware authors do not need to change their malware to make it undetectable by anti-malware software. Users of malware use obfuscation techniques to deceive anti-malware software, and new versions of the same code can be created automatically. These techniques mainly change the structure of code or change it in assembly language or byte level and do not change the functionality.

- Dead-code insertion is inserting codes that are never executed in code, for example, there can be a jump to the end of the dead code at the beginning of the dead code.

- An instruction substitution is changing the original code with instruction with the same functionality, for example, a subroutine can be in-lined.

- There are other techniques like Packing that prevent the malware to be analyzed until it is loaded to memory.

These techniques and many more prevent signature-based detection of malware. With each small-scale change, the malware should be classified again to use signature-based detection. For classification, various techniques can be used, some of those techniques are static analysis, dynamic analysis, and file entropy.

- Static analysis is analyzing software without executing, safer compared to dynamic analysis but occasionally it is useless, against code packing for example.

- Dynamic analysis is analyzing the code while executing it, regularly both dynamic analysis and static analysis are used together as hybrid analysis.

- File entropy is a factor when detecting malware since commonly malware programs have high entropy. Unfortunately, those techniques might increase the cost drastically and create a bottleneck for malware detection operations.

1.7 Behavioral Analysis Malware Detection

1.7.1 Definition of Behavioral Analysis

Behavioral analysis is a technique used in malware detection to identify and analyze the behavior of a program or file. It is a proactive approach to detecting malware that does not rely on predefined signatures or known patterns of behavior. Instead, it monitors the behavior of a program in real-time to detect any suspicious activity.

1.7.2 Definition of Behavioral Analysis

Behavioral analysis techniques are particularly useful for detecting new and unknown types of malware, as well as malware that has been modified to evade signature-based detection. This is because behavioral analysis focuses on the actions that a program takes, rather than the code it contains. This means that it can detect malware that has been modified to avoid detection by changing its code or signature.

1.7.3 Types of Behavioral Analysis

There are several types of behavioral analysis techniques used in malware detection.

1.7.3.1 Network Activity

One approach is to monitor the program's network activity, looking for suspicious connections or data transfers. For example, a malware program might attempt to connect to a known command and control server or transmit sensitive data to a remote location.

1.7.3.2 System Activity

Another approach is to monitor the program's system activity, looking for suspicious behavior such as attempts to modify system files or registry keys. Malware often needs to modify system files in order to operate, so this type of activity can be a strong indicator of malicious behavior.

1.7.3.3 Runtime Behavior

Finally, some behavioral analysis techniques involve monitoring the program's runtime behavior, looking for patterns of behavior that are characteristic of malware. For example, a malware program might repeatedly attempt to access certain system resources or perform unusual operations.

1.7.4 Sum-Up

In order to perform effective behavioral analysis, it is necessary to have a good understanding of the normal behavior of the system or program being analyzed. This can be achieved through various means, such as creating a baseline of normal behavior or using machine learning algorithms to identify patterns of behavior that are indicative of malware.

1.8 Machine Learning-Based Malware Detection

Machine learning-based malware detection does not require signatures of malware. Unlike signature-based malware detection, artificial intelligence can detect new malware. Beyond the detection of malware, they can be categorized by behavior and objectives, categorization is potentially beneficial information.

1.8.1 Variant Detection and Similarity Detection

Malware similarities analysis has slightly different versions. Variant detection is focused on variants of already known malware. Families' detection focuses on the likelihood of the sample, behavioral, structural, or structural similarities are potentially significant. Similarity detection focuses on similarities between the unclassified sample and already classified samples.

1.8.2 Steps of Malware Detection

Machine-learning-based malware detections broadly have a workflow containing five consecutive steps. The workflow steps could be designated as data extraction, preprocessing, feature selection, machine learning algorithm, and evaluation.

1.8.2.1 Data Extraction

Data extraction is the process of collecting data either by static or dynamic analysis from various sources. The scope of the analysis process depends on the objective features.

1.8.2.2 Preprocessing

Preprocessing is manipulating the extracted data before using it, commonly the manipulation aims to shape the data to be a better fit for the operation or predictably increase the quality.

- This manipulation may be data cleansing which is eliminating the logically impossible data, for example, there is a set of instances of software with a count of three characters or longer strings and the count of different bytes features if the count of different bytes is lesser than the number of three characters or longer strings of an instance, there is a high probability of false information since the standard size of a character is one byte.
- Another manipulation technique is editing the features of instances, for example, features standardized by standard scaling. Also, the elimination of noisy data instances is typically performed during preprocessing.

1.8.2.3 Feature Selection

Feature selection also known as variable selection or attribute selection is the process of reducing the dimension of instance data by selecting a relevant subset of features or transforming the feature set to a lower dimension set. The feature selection aims to increase performance by decreasing the dimension of features.

1.8.2.4 Machine Learning Algorithm

Machine learning algorithms aim to solve the classification of instances or clustering of data problems. Machine learning algorithms are categorized into three groups by the data sets they feed during the training phase. The training phase of machine learning algorithms is the pattern matching, finding, and predicting phase by the input data set, this phase is essential to perform the classification of instances or clustering of data tasks. Therefore, the training data set has a huge impact on achieving the goal.

Machine learning algorithms perform the learning phase with supervised, unsupervised, or semi-supervised training data sets.

- Supervised machine learning algorithms use datasets with labeled instances. Labeled instances definition refers to classified instances, for malware detection algorithms, the classification is essentially malware, benign, or unknown, for more comprehensive classification, families of malware provide deeper knowledge.
- Unsupervised machine learning algorithms use unlabeled training data sets to discover hidden patterns or cluster the data set without human intervention.
- Semi-supervised machine learning, also called weak supervision, combines supervised learning and unsupervised learning. That approach aims to alleviate the problem of having limited labeled data with larger unlabeled data.

1.8.2.5 Evaluation

The evaluation of machine learning depends on the success criteria, by the success metrics the performance of machine learning with the input training data set has been criticized. To sum up, the training data has a huge impact on the performance of machine learning algorithms, hence the evaluation of the algorithm matched the training data set.

1.8.3 Principal Component Analysis

The principal component analysis starts with centering the data around the origin, evaluates the center of data and relative positions of instances, then moves the instances to the relatively same position centered around the origin.

After the centering, the best-fitting linear combination is discovered. The distance between the origin and the reflection of the instance on the line of linear combination for each instance is squared and summed to determine the best-fitting linear combination. The higher sum of squared distances is the better.

After finding the best-fitting linear combination, new linear lines will be calculated for other dimensions, those new linear lines are perpendicular to the best-fitting linear combination line and like the best-fitting linear combination line, they also pass through the origin which is the center of the data graph at that stage.

For each dimension (or features in feature vectors of the implementation), the square of the distance between the reflections of data on the new linear line and the origin is calculated. The dimensions with bigger calculated scores are better-fit and the best candidates are selected among the scores. Only the selected dimensions are used during the rest of the execution.

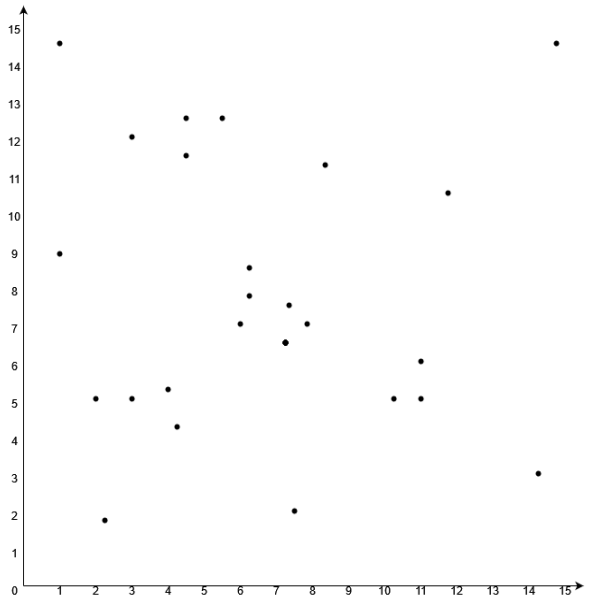
Chapter 2
Algorithm Description

In this chapter, we propose a new unsupervised instance selection algorithm. The complete codes are provided in the GitHub repository [7]. The Nearest Cluster Enemy (NCE) instance selection algorithm works in four steps.

1. The first step is preprocessing. Preprocessing reduces the number of features in the feature vector of the instances from the data set and scales the data of the feature vector.
2. The second step is clustering. Clustering prepares the data set for the third step and further to the fourth step. So the clustering parameters are parameters of NCE.
3. The third step is searching for the nearest enemy of each instance with respect to clusters. The nearest enemy with respect to clusters is the nearest instance from a different cluster set of the subject instance.
4. The last step is elimination. Two different elimination methods are proposed for the last step. Both methods depend on the result of the second step, and the nearest enemy with respect to clusters.

For each step of the algorithm, the result of the step is represented as a graph in relative sections. The graph of the original (non-reduced) data set used for the demonstration of particular steps of NCE is presented in Figure 2.1.

Figure 2.1 Original Graph



2.1 Preprocessing and Feature Extraction

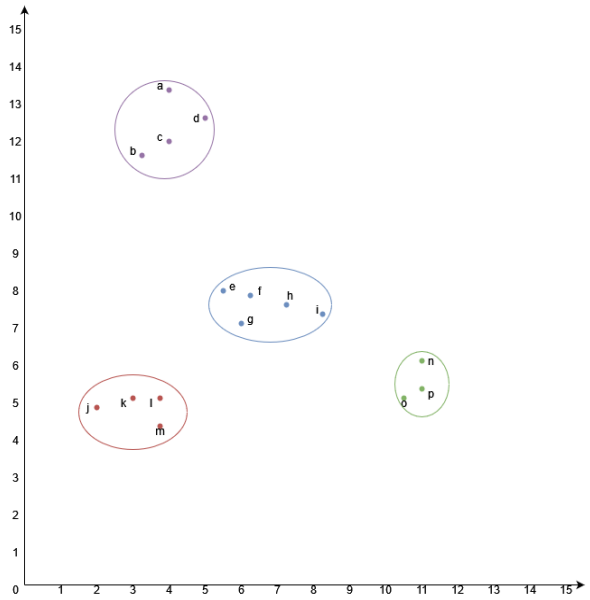
The first step is preprocessing, for all steps of the NCE algorithm work depending on the feature vector of instances. The feature vector of instances is a numeric representation of numbers of bytes as mentioned in the first chapter. There are over 2000 features in each instance of EMBER [8] data set instances. To preprocess the data set, principal component analysis has been used. The purpose of this analysis is to increase the accuracy of the model and decrease time complexity by extracting the most significant features. Moreover, feature vectors are scaled by using the StandardScaler [9] function of sklearn.preprocessing library because scaling the data makes it easier to process and makes extreme data processable.

2.2 Clustering

The Second step is clustering. Clustering is essential since the third and last step depends on clustering labels. Clustering labels are the numeric representation of clusters, unique for each cluster. Density-based clustering (DBSCAN) has been used during this step because the number of clustering should not be restricted.

As depicted in Figure 2.2, certain instances are not displayed and, consequently, will not be used in the rest of the steps since they are not clustered.

Figure 2.2 Clustering Result Graph

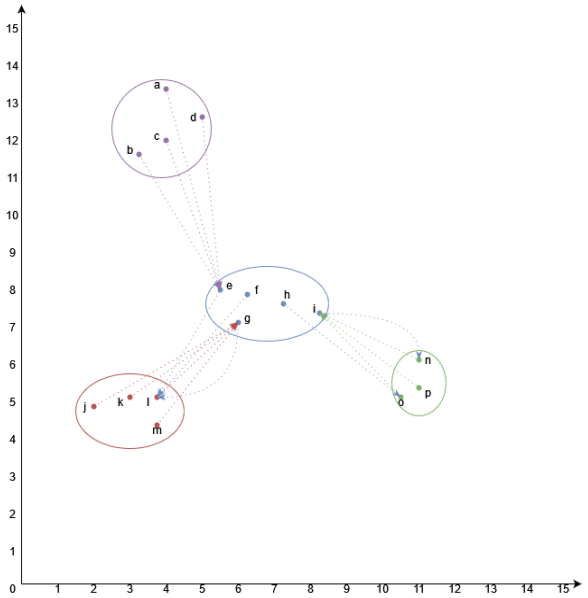


2.3 Detection of Nearest Enemies

The third step is the detection of the nearest enemy with respect to clustering. For each instance, the nearest enemy with respect to clustering is found. The third step has been done by brute force. Therefore, the time complexity of the third step is relatively high compared to the first, second, and fourth steps. Suggestions to decrease expected execution time are presented in the implementation notes section. The distance between two instances is Euclidean distance with respect to feature vectors of instances.

Figure 2.3 shows the clustered data set and their nearest enemies with respect to clustering. For each instance, the tip of the arrow points to its nearest enemy with respect to clustering.

Figure 2.3 Detection of Nearest Enemies Result Graph

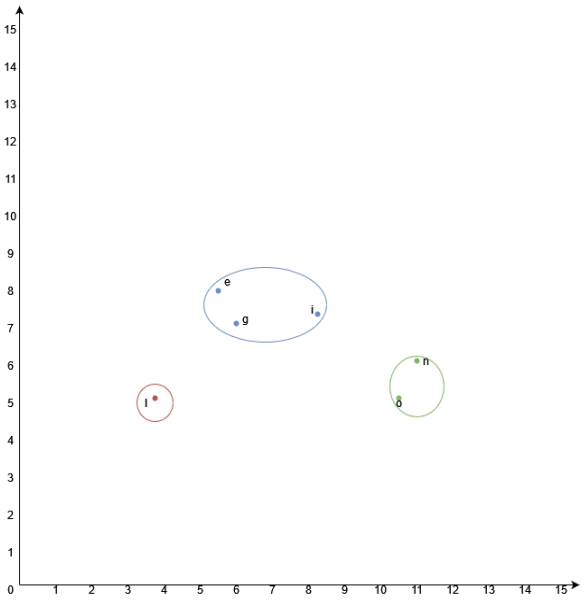


2.4 Elimination

The last step is elimination. Two eliminating techniques are proposed in this step. In the first technique, the nearest enemy with respect to the clustering of each instance is detected and consequently, the instances undetected as the nearest enemy are eliminated.

Figure 2.4 shows the result of the first elimination technique NCE 1.

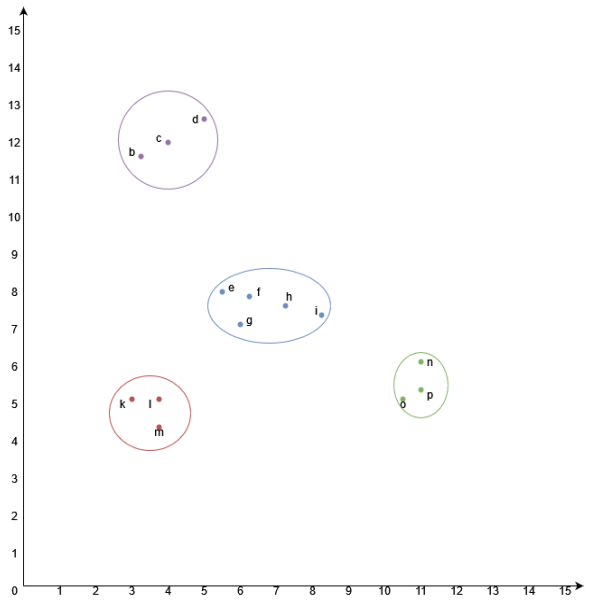
Figure 2.4 Elimination 1 Result Graph



In the second technique, to determine if an individual instance X should be eliminated, the algorithm calculates the distance between X and its nearest enemy NE and compares it to the distances between the other members of the cluster and NE. If the number of cluster members that are closer to NE than X exceeds the threshold, X is eliminated from consideration.

Figure 2.5 shows the result of the second elimination technique NCE 2.

Figure 2.5 Elimination 2 Result Graph



Chapter 3
Implementation notes

3.1 Instance PE Class

The data set is kept as an object array in the implementation. The class has 5 fields.

Two of the fields of the class contain just holding information from the data set, they are the SHA256 and the feature vector of the instance. The SHA256 is the digital signature of the software instance, created by the Secure Hash Algorithm SHA256. The feature vector is numerically the representation of features of instances as mentioned in the Algorithm Description section.

The three other fields are the cluster label of the instance and its position in the array, and the SHA256 of the nearest enemy with respect to the clustering.

3.2 Preprocessing and Clustering

The `sklearn` [10] library of Python provides preprocessing and clustering functions. The `sklearn` library has functionalities about supervised learning, unsupervised learning, model selection and evaluation, inspection, dataset visualization, dataset transformation, and dataset loading utilities, and it is widely used in the machine learning field.

3.2.1 Principal Component Analysis

Principal component analysis (PCA) can be imported from `sklearn.decomposition` and perform principal component analysis. PCA function of `sklearn` library has ten arguments `n_components` is the number of components to be kept (feature vector in PE object), the data to be analyzed, `copy`, `whiten`, `svd_solver`, `tol`, `iterated_power`, `n_oversamples`, `power_iteration_normalizer`, and `random_state`.

A more detailed description of arguments can be found in the documentation [11].

In the implementation, all the arguments kept default except `n_components` and the data, the `n_components` argument is set as fifty. The reason for using fifty as the argument is to keep the execution time small while not neglecting a high amount of data.

3.2.2 Standard Scale

StandardScaler can be imported from `sklearn.preprocessing` [10] and perform scaling. `StandardScaler` [9] function has four arguments, one of which is compulsory and three are optional. The required

argument is the data to be scaled. To perform the scale, the scale score z is calculated as:

$z = \frac{x-u}{s}$

, x is the data sample, u is the mean of the training samples, and s is the standard deviation of training samples. The standard deviation is a measure of the amount of variation in a dataset.

The optional arguments of `StandardScaler` are `copy`, `with_mean`, and `with_std`. All of these optional arguments have a default value of `True`. If the `copy` argument is set to `False`, the function tries to avoid copying the data and operates scaling in place. The data set is centered if the `with_mean` argument is `True`. In other words, if `with_mean` is `False`, then u is zero. If the `with_std` argument is `True`, the data is scaled to have unit standard deviation. Consequently, if the `with_std` argument is `False`, then s is one, and the standard deviation has no impact on the calculation of the scale score z.

3.2.3 Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) can be imported from sklearn.cluster and performs density-based clustering. Density-based clustering depends on two parameters minimum sample number and epsilon. Epsilon is the maximum Euclidean distance between instances in a cluster. The minimum sample number is the minimum number of instances to form a cluster.

The algorithm works as, clustering starts with a random instance, every instance in the epsilon range included in the group to create the cluster, then this is repeated for each new group member recursively when there is no instance left in the epsilon range, group members counted if the count is smaller than the minimum sample number, corresponding instances are considered noise if the count is equal to or greater than the minimum sample number, a new cluster is formed.

The search for the nearest enemy with respect to the clustering is done by brute force. For each individual instance, the distance between the instance and all other instances is calculated. Calculation of the distance is done by using the `dist()` function imported from the `math` library in Python. `math.dist()` function requires 2 inputs, the array of positions of two points, and returns Euclidean distance between the points.

3.3 Elimination

In the implementation, these points are feature vectors of PE objects, and these feature vectors are 50-dimensional arrays. While calculating Euclidean distances between instances nearest instance from different cluster sets is recorded. This process can be easily parallelized.

Therefore, in implementation, the `Pool` function from the `multiprocessing` library of Python is used to create a pool of processes that later will execute a parallelized function. The `Pool` function requires one input, to use the full available capacity of the machine number of all available CPU cores can be detected by using `cpu_count()`. Afterward, `apply_async()` is called to create parallel asynchronous tasks. Parallel tasks are executions by different CPUs at the same time.

Asynchronous means those tasks do not block each other. If parallel tasks are not asynchronous, they have a synchronization mechanism and CPU cores follow this synchronization mechanism to start a new task. On the other hand, asynchronous parallel tasks start to execute the new task immediately after they finish the current task until task pool is empty (no task left to execute).

`apply_async()` requires three arguments, the function to be executed, arguments of the function, and the callback function. The callback function catches the results of each task and processes it accordingly.

In the implementation, the executed function does the brute-force search of the nearest enemy according to clustering and records the nearest enemies' SHA256 and index number to the corresponding fields of the instance object, arguments change correspondingly one by one for each instance and the callback function builds an array collecting return of NE functions which are instances with information of their nearest enemy with respect to clustering.

3.3.1 Elimination Technique 1

The first elimination technique is the elimination of all instances which are not the nearest enemy of another instance from a different cluster, that process has the risk of taking an exhaustive amount of time. Because iteratively searching for each instance to mark their nearest enemy has a higher time complexity compared to the dynamic programming approach. To prevent exhaustive search operations, time complexity can be transformed into space complexity in that elimination technique. An array size equal to the object array m created filled with zeros as initial values, in a loop, each nearest enemy's index can be marked with one in m since the index number of each enemy is recorded in the third step, then only the objects corresponding to marked indexes of m are kept, unmarked objects are eliminated. The Pseudo code of this technique is represented in Algorithm 1:

Algorithm 1 Eliminating Technique 1

1: $m[\text{length}(Object\_list)] = \{0\}$ {m is an array with size of Object_list, all elements initialized as zero}
2: M {M is the array of all PE objects}
3: **for** m in M **do**
4: $m[Object\_list.NE] = 1$ {Object_list.NE is the index number of the nearest enemy}
5: **end for**
6: **for** i in $range(0, \text{length}(M))$ **do**
7: **if** $M[i]$ **then**
8: M is not eliminated
9: **else**
10: M is eliminated
11: **end if**
12: **end for**

3.3.2 Elimination Technique 2

The second elimination technique eliminates by comparison of the distance between an object Obj and its nearest enemy against the distance between the objects from the same cluster and object Obj's nearest enemy. To shrink the search space per object, first, a two-dimensional object array is created, and arrays of clusters are created. Arrays of clusters are object arrays that are part of the same cluster. To decrease expected time complexity instead of searching for each cluster's elements, an empty two-dimensional array with the first dimension's length equal to the length of the object array and appended the list correspondingly in one loop with iteration equal to the length of the object. After creating the two-dimensional array of clusters, the elimination function is called in an asynchronous parallel loop. The arguments for the elimination function are an object, the array of the cluster, and the list of objects. By the cluster label field of the object Obj, the cluster of Obj is input for the function not the whole 2-dimensional array of clusters is passed as argument. During each iteration, the first distance between the object Obj and its enemy is computed for further comparisons, then, the distance between each object in the cluster and the nearest enemy of object Obj is calculated and compared with the distance

of the object Obj. Consequently, if the number of objects closer to the nearest enemy of the object Obj compared to the object Obj is equal to or more than the threshold, the object Obj is eliminated. The threshold for this function is equal to the minimum sample number described in the clustering step. The Pseudo code of the function is represented in Algorithm 2:

Algorithm 2 Elimination Technique 2

```
1: cluster_array {The 2D array of all clusters with instances of each cluster}
2: Obj_array {Obj_array is the array of all PE objects}
3: minimum_samples {Minimum sample number that is used in clustering}
4: for Obj in Obj_array do
5:    counter ← 0
6:    Obj_distance ← distance(Obj, Object_list[Obj.NE])
7:    for c_obj in cluster_array[Obj.cluster] do
8:       if Obj_distance > distance(c_obj, Object_list[Obj.NE]) then
9:          counter ← counter + 1
10:      end if
11:   end for
12:   if minimum_samples > counter then
13:      Obj is not eliminated
14:   else
15:      Obj is eliminated
16:   end if
17: end for
```

Chapter 4

Test Background Information

4.1 Evaluation

Instance selection is a process of selecting a subset of instances from a large dataset that can be used to train a machine learning model efficiently. It is often used to remove irrelevant or redundant instances from the dataset, which can improve the performance of the learning algorithm and reduce the computational cost. Instance selection is especially useful when dealing with large and complex datasets that are time-consuming and resource-intensive to process.

The primary objective of instance selection is to select a representative subset of instances that can maintain the predictive power of the learning algorithm while reducing the size of the dataset. The selected instances should be informative enough to capture the essential characteristics of the original dataset while eliminating noisy or redundant instances that may adversely affect the performance of the algorithm.

The performance evaluation of instance selection algorithms is crucial in determining their suitability for different applications. The evaluation is typically based on three metrics: reduction rate, accuracy, and computational time.

4.1.1 Reduction Rate

Instance selection algorithms are widely used in machine learning to reduce the size of large datasets, without compromising their predictive performance. The reduction rate is a crucial metric that measures the effectiveness of an instance selection algorithm in reducing the size of a dataset.

The reduction rate is defined as the ratio of the number of instances in the reduced dataset to the number of instances in the original dataset. A low reduction rate implies that the algorithm effectively removed redundant and irrelevant instances from the dataset, leading to a smaller and more manageable dataset. This, in turn, can improve the efficiency of the learning algorithm, as it reduces the number of instances that need to be processed.

However, it is important to note that a high reduction rate does not necessarily guarantee good performance. The instance selection algorithm should also ensure that the selected instances are representative of the original dataset and maintain its predictive power. Therefore, it is necessary to evaluate the reduction rate in conjunction with other metrics such as accuracy and computational time.

4.1.2 Accuracy

In machine learning, the goal is to create models that can accurately predict outcomes based on input data. The accuracy of the model is directly influenced by the quality of the dataset it is trained on. Therefore, instance selection algorithms aim to select a subset of instances that are representative of the original dataset and can maintain the accuracy of the model.

Internal methods assess the quality of the clusters formed by the instance selection algorithm. These methods rely on measures such as intra-cluster similarity and inter-cluster dissimilarity to evaluate the clustering results. The higher the intra-cluster similarity and the lower the inter-cluster dissimilarity, the better the quality of the clusters.

External methods, on the other hand, compare the clustering results to the original labels of the dataset. The quality of the clustering is evaluated by metrics such as precision. A good instance selection algorithm should achieve high precision and recall values, indicating that it correctly identifies instances that belong to the same class and removes instances that do not belong to the class.

4.1.3 Computational Time

In practical applications, the computational time of an instance selection algorithm can be a critical factor that determines its feasibility and usefulness. An algorithm that takes a long time to process large datasets can be impractical and inefficient for real-world scenarios. Therefore, a computationally efficient instance selection algorithm that can achieve a high reduction rate and accuracy is desirable.

The computational time of the algorithm can be affected by various factors such as the size and complexity of the dataset, the selection criteria used by the algorithm, and the implementation of the algorithm itself. For example, an algorithm that employs computationally intensive selection criteria, such as distance-based or clustering-based methods, may take longer to process than simpler algorithms that use heuristics or random sampling. Similarly, even though high-level languages such as Python have a wide range of libraries and community support, an algorithm that is implemented using a high-level programming language may be slower than one that uses lower-level languages or specialized libraries for optimized computations.

4.2 Endgame Malware BEnchmark for Research Dataset

The Endgame Malware BEnchmark for Research (EMBER) dataset is a publicly available dataset that was created for the purpose of training and testing malware-detecting machine learning models. It contains features extracted from 1.1 million binary files that were taken from Windows portable executables. Out of these 1.1 million files, 900,000 instances were intended to be used as training data, while the remaining instances were intended to be used as testing data.

The primary objective of the training data is to feed the artificial intelligence (AI) during the learning process. In contrast, the testing data is used to evaluate the accuracy of the AI after the learning process has been completed. The goal of the EMBER dataset is to provide a comprehensive, diverse, and realistic dataset that can be used to evaluate the performance of malware detection algorithms. It includes a wide range of malicious and benign files, with varying levels of complexity and sophistication.

The EMBER dataset has several advantages. First, it is publicly available, making it accessible to researchers and practitioners worldwide. Second, it is large and diverse, which allows for a comprehensive evaluation of different malware detection algorithms. Third, it includes features that are extracted directly from the binary files, which allows for a more accurate representation of the data and a better evaluation of the algorithms. Finally, it includes both training and

testing data, which allows for a comprehensive evaluation of the performance of the algorithms both during and after the learning process.

For more comprehensive details, you can take a look at the paper EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. [8]

4.2.1 Windows Portable Executable

The Windows portable executable (PE) is a file format that was introduced by Microsoft for executables in the Windows operating system. It is a common file format used for applications, dynamic-link library (.dll) files, audio compression manager (.acm) files, DirectShow (.ax) files, system files (.sys), Windows ActiveX Control files (.ocx), object, object code files, and many more different types of Windows files.

PE files contain the necessary information for the Windows Operating System loader to manage the wrapped executable code. This information includes the code and data sections of the executable, as well as the metadata and resources associated with the executable. The metadata includes information about the file, such as its version number and timestamp, while the resources can include icons, dialogs, and other embedded files.

PE files are essential to the operation of Windows and are used by the operating system to load and execute code. They also play a critical role in the security of the Windows operating system, as they can be used to package malware and other malicious code. As a result, understanding the format and contents of PE files is essential for developing and implementing effective malware detection and prevention strategies.

4.2.2 Library to Instrument Executable Formats

The LIEF library is an open-source project that offers a library to parse, modify, and abstract executable file formats such as Executable and Linkable Format (ELF), Portable Executable (PE), and Mach Object (Mach-O). It provides a set of functionalities to read and analyze the structure of the executable files, extract features, and manipulate the files in various ways. LIEF is cross-platform and supports three different programming languages: Python, C, and C++.

The source code to extract features of PE files from the EMBER dataset is published on GitHub together with the dataset, and it uses the LIEF library. The feature extraction process involves parsing the PE file structure and extracting various properties such as file header, section headers, imports, exports, resources, and strings. These features are then used as input to train and test machine learning models for malware detection.

The availability of the source code for feature extraction enables researchers and practitioners to replicate the feature extraction process and apply it to other datasets or use it for other purposes. Additionally, the use of a widely available and well-documented library like LIEF simplifies the implementation of feature extraction and allows for easier integration into existing projects.

4.2.3 Format of EMBER Dataset

The EMBER dataset, which contains features of 1.1 million binary files extracted from Windows portable executables, stores the extracted data in a JSONL file format. JSONL files differ from JSON files in that each line in the file contains a single JSON object, with new line characters separating each object. Therefore, each row in the EMBER dataset corresponds to a different PE file, and contains various features of the file in JSON format. The features extracted from the PE files are organized into different categories, including ”general”, ”header”, ”imports”, ”exports”, ”section”, ”histogram”, ”byteentropy”, and ”strings”

- “sha-256” is hash of the PE file by sha-256 hashing algorithm.

- "appeared" is the year and month of appearing.
- "label" is the classification of PE file as benign, malicious, or unlabeled.
- "general" is set of general file information features includes size, virtual size, number of imported functions, number of exported functions.
- "header" consists of the timestamp, target machine and a list of image characteristics.
- "imports" consists of imported functions and their libraries parsed from import address table. The import address tables are part of Windows executables and DLLs which contain addresses of imported functions.
- "exports" is the list of exported functions.
- "section" contains properties of each section.
- "histogram" is the list of the count of each byte.
- "byteentropy" is the approximate joint distribution of entropies and byte values. "strings" is the list of strings at least five characters long and their characteristics.

During evaluation "sha-256" and "histogram" were used. Each step of the NCE algorithm uses the "histogram" list as a feature array and the hash of the PE file is kept for backtracking the instance since during the first step of the NCE algorithm, the original feature array is reduced by PCA and scaled. "Histogram" has been used for instance selection because counting the occurrence of each byte in a file can be used as a feature in machine learning models for malware detection.

- By analyzing the distribution of byte counts across a large number of files, machine-learning models can learn to identify patterns that are indicative of malware. For example, some malware may contain a large number of specific bytes, such as null bytes (0x00) or escape characters (0x1B), that are not commonly found in legitimate files.
- By detecting an unusually high frequency of these bytes in a file, machine learning models can flag the file as potentially malicious. However, counting the occurrence of each byte alone may not be sufficient for effective malware detection, as many legitimate files may also contain uncommon byte values.

Therefore, byte counts are often used in conjunction with other features, such as byte n-grams, or file metadata to create a more comprehensive malware detection system. [12][13]

4.3 Silhouette Method

The Silhouette method is an internal evaluation technique for clustering results. Internal evaluation means the evaluation of results based on the data itself. The Silhouette method has been used to evaluate the quality of instance selection by the clusters. The Silhouette is a method to measure the quality of clusters, since the algorithm works with unlabeled data, using the Silhouette method is consistent to judge the quality of the clustered data sets' formation. That method yields a floating point which is called the Silhouette distance or coefficient. The coefficient is between minus one and one. If the score is close to one, the method indicates clusters are accurately formed, on the other hand near minus one means the opposite. Calculation of the Silhouette coefficient depends on two characteristics of the clusters. Cohesion is characteristic of the similarity of instances that share the same clustering set and separation is characteristic of the dissimilarity of instances from different clustering sets.

1. To calculate the Silhouette coefficient, the first step is to calculate the mean of the intra-cluster distance a. For each member of the cluster, the mean distance between the member and all other instances that are sharing the same cluster is calculated.

2. In the second step, the mean nearest distinct cluster distance b is calculated. The mean distance between the member and instances of a distinct cluster set is calculated for each distinct cluster, and then the distinct cluster with minimum mean distance is selected as the nearest distinct cluster and the mean distance as b. These a and b values are calculated for each instance.

3. As the last step, the Silhouette coefficient for one instance is calculated with the formula (b - a) / max(a, b). The result of the Silhouette method is the average of the Silhouette coefficient of whole instances.

In Python, this method is named "silhouette_score" and is publicly available to import from the "sklearn.metrics" library. "silhouette_score" function has 6 input arguments, two of which are compulsory, the instance data and the cluster labels for each instance. The other three are optional to fill because they have default values.

- One of these three methods is the "metric" which is the metric to calculate the distance between instances and is set Euclidean distance as default.

- Another one is "sample_size", it is optional to determine the size of the sample while using a random subset of the data, it is set as none in the default which means no sampling is used.

- The other argument is "random_state" and it is a seed for random number generation of the subset of samples, if "sample_size" is none "random_state" is not used, the default is none.

- The last argument is "kwds". "kwds" does not have a default value, completely optional, it stands for optional keyword parameters, and these parameters are directly passed to the distance function. During the evaluation, only compulsory parameters are used.

4.4 Clean Form

In the case of instance selection, the goal is to select a subset of instances that are representative of the original data and can maintain the predictive power of the learning algorithm. One issue that may arise during the instance selection process is the presence of noisy data, which can negatively impact the quality of the selected instances. To address this issue, a first step is taken to purge the dataset of noisy data. The number of instances that do not involve a cluster are identified as noise and removed from the dataset.

After the noisy data is purged, the remaining data is clustered using density-based clustering techniques such as DBSCAN. The Silhouette coefficient is then calculated to evaluate the quality of the clustering results. The Silhouette coefficient is a metric used to assess the quality of a clustering solution. It measures how similar an instance is to its own cluster compared to other clusters. A high Silhouette coefficient indicates that the instances are well-clustered and that the clustering solution is of good quality.

The clean form refers to the dataset without the instances that are not part of a cluster. This is because density-based clustering algorithms like DBSCAN require a minimum cluster size, and instances that do not belong to any cluster are assigned a cluster label of -1. The clean form of the dataset is the preprocessed training dataset where all instances are part of a cluster and can be used for model training.

4.5 Purity

The purity of the clustering set is a measure of how well the clusters represent the different classes of data. It is an external evaluation technique, which means that it requires knowledge of the ground truth or the actual class labels of the data. The purity of a cluster is defined as the proportion of instances in the cluster that belongs to the most frequent class in that cluster.

To calculate the purity of a clustering set, the class labels of the instances in each cluster are determined. For each cluster, the class label that appears most frequently in the cluster is identified as the predominant class for that cluster. The number of instances in the cluster that belongs to the predominant class is then counted, and the sum of these counts across all clusters is calculated. Finally, this sum is divided by the total number of instances in the dataset to obtain the purity of the clustering set.

A high purity score indicates that the clusters are well separated and contain instances belonging to a single class, while a low score suggests that the clusters are mixed and contain instances from multiple classes. Purity is a useful evaluation metric for clustering algorithms, especially in scenarios where the ground truth is known. However, it is important to note that in many real-world applications, the ground truth may not be available, making it difficult to use purity as an evaluation metric.

4.6 Hyperparameter

Hyperparameter optimization, also known as hyperparameter tuning, is an essential step in machine learning. It involves selecting the optimal parameters for the machine learning algorithm that will yield the best performance on the given dataset.

In the context of the NCE algorithm, hyperparameter optimization refers to tuning the parameters of the density-based clustering algorithm used in the third and fourth steps of the algorithm. The performance of these steps depends on the clustering labels of the input data, as the nearest enemy of instances according to clustering changes by clustering labels.

The density-based clustering algorithm used in the NCE algorithm has two important parameters: the minimum number of instances in a cluster and the maximum distance between cluster members. These parameters are explained in the implementation notes, and they are tuned during hyperparameter optimization.

To evaluate the performance of the NCE algorithm with different parameters of density-based clustering, the results are compared and analyzed on a test dataset. The goal is to find the optimal hyperparameters that will result in the best performance of the NCE algorithm on the given dataset. results.

4.7 Stratification

Stratification is a technique that is widely used in data science and machine learning to divide a large dataset into distinct groups or layers, with the purpose of processing each group individually and then combining the results to form the final result. The goal of this technique is to reduce the analysis time, as the time complexity of many algorithms depends on the size of the dataset.

The stratification technique is particularly useful when analyzing large datasets that cannot be processed by a single machine or in a single pass. By dividing the dataset into smaller, more manageable subsets, it is possible to distribute the processing across multiple machines, or to process the subsets sequentially on a single machine. This can result in significant reductions in the time required to complete the analysis.

The stratification technique can be applied in various ways, depending on the nature of the data and the requirements of the analysis. One common approach is to randomly partition the dataset into subsets of equal size, with each subset containing a representative sample of the

overall data. Another approach is to partition the data according to some relevant characteristic, such as geographic region, time period, or demographic group.

One potential drawback of the stratification technique is that it may decrease the accuracy or quality of the final result. This is because the subsets of the data may not be representative of the whole dataset, and may not capture all of the relevant information. However, this drawback can be mitigated by careful selection of the subsets, or by combining the results in a way that takes into account the differences between the subsets.

For instance, let's say we have a dataset with n size to be analyzed. A loop iterates for each member of the array, and in each iteration, another loop iteration for each member of the array takes one second. If the array size is 60, the total execution time will be 3600 seconds (1 hour). On the other hand, if the stratification technique is used, and same-sized data sets are grouped from the original data set, there will be two executions with halved-sized arrays each, resulting in two executions lasting up to 900 seconds (15 minutes). In that brief example, the total execution time is halved by using stratification.

4.8 Nimble Instance Selection

Nimble Instance Selection (NIS) [14] is an unsupervised instance selection algorithm. This algorithm is based on hyper-rectangles, which are generalized parallelograms with higher dimensions. Hyperrectangles are formed by a sequence of consecutive rectangles in two-dimensional space, and NIS creates conjectural hyperrectangles indirectly, by forming consecutive rectangles and keeping only one random instance in each rectangle.

The goal of NIS is to reduce the size of the dataset while retaining its representativeness so that it can be used in downstream tasks such as clustering, classification, or intrusion detection. The algorithm achieves this goal by selecting a subset of the most informative instances while discarding the less informative ones. The resulting subset is smaller and more manageable, while still retaining most of the relevant information.

One of the advantages of NIS is its speed and scalability. Since the algorithm does not calculate the exact coordinates of the hyperrectangles, but instead forms them indirectly, it can process large datasets efficiently. Additionally, the algorithm is unsupervised, which means that it does not require labeled data, making it suitable for a wide range of applications.

However, a potential drawback of NIS is that it may discard some important instances, which can result in a loss of information or accuracy. To mitigate this drawback, it is important to carefully choose the parameters of the algorithm and to evaluate the quality of the resulting subset using appropriate metrics. Nevertheless, NIS has shown promising results in a variety of applications, such as image classification, text classification, and sensor data analysis.

A pseudo-code of the NIS algorithm is represented in Algorithm 3:

Algorithm 3 NIS algorithm

Require: Data set X, scaling parameter α (default value is 1.0)
Ensure: Indices of the unique rows in the transformed data set
$u \leftarrow$ Minimum of each column of X
$transformedX \leftarrow X - u$
$transformedX \leftarrow \alpha * transformedX$
$v \leftarrow$ Standard deviation of each column of X
for each column i in X **do**
  $transformedX[:, i] \leftarrow transformedX[:, i]/v[i]$
end for
$transformedX \leftarrow$ Round each element of $transformedX$ to the nearest whole number
$transformedX \leftarrow$ Replace NaN values in $transformedX$ with zero
$transformedX, indices \leftarrow$ Unique rows and their indices in $transformedX$

return indices

4.9 K-Nearest Neighbors Classification

K-nearest neighbors (K-NN) classification is a supervised learning algorithm commonly used for classification tasks. It is a simple and effective algorithm that can be applied to both binary and multiclass classification problems.

In K-NN classification, each instance in the dataset is represented by a set of features or attributes. The class of each instance is also known. When a new instance is presented to the algorithm, its class is determined based on the class of its k-nearest neighbors.

The k value is a user-defined hyperparameter that specifies the number of neighbors to consider. Typically, a larger k value will result in a smoother decision boundary, while a smaller k value may lead to overfitting.

To classify a new instance, the distance between the new instance and every instance in the dataset is calculated. The distance can be measured using various distance metrics, such as Euclidean distance, Manhattan distance, or Minkowski distance. The k nearest neighbors are then identified based on their distance to the new instance. The class of the new instance is determined by taking the majority class of the k-nearest neighbors.

K-NN is a simple and intuitive algorithm, but it has some limitations. One of the main challenges of K-NN is choosing the appropriate value of k. If k is too small, the algorithm may be sensitive to noise in the data and may not capture the underlying structure of the data. On the other hand, if k is too large, the algorithm may be too generalized and may not capture the local structure of the data. Additionally, the algorithm can become computationally expensive when the dataset is large, as the distance between every instance in the dataset needs to be calculated for each new instance.

Chapter 5

Test Implementation Notes

5.1 Hyperparameters

During testing of the NCE algorithm, various argument pairs were used to evaluate the algorithm's performance with different hyperparameters. The argument pairs used in the tests were combinations of minimum cluster size and maximum distance between instances, epsilon, with values of 0.0005, 0.001, and 0.01 for epsilon, and 3 and 5 for minimum cluster size.

The reason for using these specific epsilon values was that larger values cover a larger area for scaled instance coordinates, which increases the probability of the sizes of the clusters being very large, resulting in decreased performance of NCE. On the other hand, smaller epsilon values yield a result data set with a reduction rate of only 0.2%, which is an insufficient amount of data for evaluation.

Minimum cluster sizes of 3 and 5 were used because higher values than 5 reduced the data set excessively, resulting in an insufficient amount of data for evaluation, and smaller numbers yielded a low reduction rate. However, there was one exceptional pair with a minimum cluster size of 2 and epsilon of 0.0001, which was a trial of balancing the reduction rate by stretching extremes of clustering. The results of this exceptional case are enclosed in the Testing Results section.

It should be noted that experiments with different pair options than the proposed pairs have a chance of incrementing the performance of the algorithm, but excessive trials have a chance of resulting in over-fit results. Therefore, the proposed pairs were used to maintain a balance between testing the algorithm's performance with different hyperparameters and avoiding overfitting.

5.2 Training and Testing Data Sets

In the EMBER data set, there are six training data sets and one testing data set. However, in the tests conducted for evaluating the NCE algorithm's performance in terms of Silhouette coefficient calculation and purity test, only one testing data set has been used.

This decision has been made due to the high computational cost of running the NCE algorithm on the entire EMBER data set. With 900 thousand instances, it takes more than three days (72 hours) to yield results, while using 170 thousand instances reduces the yielding time to approximately between one and thirteen hours. Therefore, in order to meet deadlines and optimize the testing process, only one training data set, specifically training data set number one, has been used in the Silhouette coefficient calculation and the purity test.

Although this limitation may have some impact on the generalization ability of the results, it is still a reasonable approach given the time constraints and computational resources available. Overall, the decision to use only one testing data set and one training data set for Silhouette coefficient calculation and purity test in the NCE algorithm testing has been made with consideration of the trade-off between computational cost and time constraints.

5.3 Working With NIS Algorithm

The result of an unlabeled data set provides more capability for the evaluation techniques for the result. Beyond the evaluation, the clustering of unlabeled data provides information on the alignment of instances for pattern matching. Therefore, clustering of the unlabeled dataset is desirable information, and the existence of that knowledge brings various options for evaluation. The NIS algorithm selects the instances without grouping or clustering the data set, hence, to be able to calculate the Silhouette coefficient and perform the purity test, two different approaches are used clustering after the NIS algorithm NIS_a, and clustering before the NIS algorithm NIS_b.

- NIS_a is performed using the NIS instance selection algorithm and then using the clustering algorithm with the same input argument pairs of the NCE instance selection algorithm. The resulting clustered data is used for evaluation by the Silhouette coefficient and purity test.

- NIS_b is performed using the NIS instance selection algorithm to an already clustered instance data set with the same input argument pairs of the NCE instance selection algorithm. And the reduction rate of the NIS algorithm and clustering kept similar to the NCE algorithm to compare the results of the tests by similar reduction rates.

5.4 Silhouette Coefficient

In Python, the function to calculate Silhouette coefficients exists within the Metrics class of the Sklearn library. The labels and the PE objects' positions (the feature arrays) are put in two arrays, appended in order. The label array, the position array, and the Euclidean option for metric are input arguments to the Silhouette coefficient function. The silhouette coefficient of the input data set was returned as a result. For each input pair of epsilon and minimum cluster size, the results of the NCE and the NIS algorithms are represented "Test Results" section.

5.5 Purity

A purity test of the result needs ground truth of the classification of instances. And original classifications of instances are not kept during the execution of the algorithm. As a first step, original classifications of instances are restored. To restore original classification labels, hash strings of the result of the algorithm are formed into a string array.

For each instance, the hash string of instances of the original data set is checked if they are in the string array. When an instance is detected by its hash string, it is appended into a dictionary of the hash string of the instance and the original classification of the instance. In a two-dimensional PE object array, the clusters are grouped into the index of their array, then for each cluster group in the two-dimensional array, the number of benign and malicious classified instances are counted, and the count of bigger classification is added to the total. And the result of the test is the total divided by the size of the result of the tested algorithm.

5.6 K-Nearest Neighbors Classification

In the implementation of the KNN classification test, the clustering label of instances is not used. Instead, hash strings of instances in the result of the algorithm are used to recover the ground truth classification labels. Once the labels are recovered, for each instance in the testing dataset, the k-nearest instances are discovered based on their Euclidean distance. The value of k is chosen to be odd to avoid even cases, and the instance is classified as the label of the majority of its k nearest neighbors.

To perform KNN classification in Python, the Scikit-learn library provides a KNeighborsClassifier class. The number of neighbors, denoted as "n_neighbors", can be set to the desired value to change the "N". The location of training data set instances (feature vector) and their labels are passed to the class as arguments of the "fit" method to train the KNN classifier. Then, the "predict" method is executed with the testing data set as an argument to return the predicted labels of the input testing data set.

To avoid overfitting, experiments of the KNN classifier were performed by using different n values, such as 5, 9, 13, and 17. Smaller values were not used to avoid overly-fit results, and larger values were avoided to prevent over-generalization. The Scikit-learn library provides flexibility in the choice of distance metrics and other parameters that can be tuned to improve the performance of the KNN algorithm. [15]

Chapter 6
Test Results

6.1 Purity and Silhouette Coefficient

- Original: This refers to the results obtained by clustering the original training data set using the density clustering algorithm.
- NCE 1: This refers to the result of the NCE (Nearest Cluster Enemy) algorithm with the first elimination technique. In this technique, only the nearest enemy of other instances is kept.
- NCE 2 is another variant of the Nearest Cluster Enemy algorithm that uses a second elimination technique. In this technique, the algorithm identifies the instance that has instances from the same cluster that are closer to its nearest enemy than the size of minimum samples of the clusters (the argument used in clustering). The instance is then eliminated from the data set. The threshold value is determined based on the minimum cluster size.
- NIS_a refers to the result of the NIS algorithm clustered after performing the NIS algorithm.
- NIS_b algorithm refers to the result of the NIS algorithm performed on performing the already clustered data set.

The size of the original training data set is the first training test of EMBER (170 thousand instances), which means that all the clustering algorithms were applied to this data set. The tables are used to compare the results of the different clustering algorithms and to determine which algorithm produces the better clustering results.

6.1.1 Silhouette Score Results

Table 6.1 displays the Silhouette score of the clustering results obtained from the different algorithms with specific clustering arguments. However, it is important to note that the Silhouette method is not applicable to the result of NIS_a. This is since the NIS algorithm does not cluster the data properly using density-based clustering, and only a small percentage of the data set (less than one percent) is either clustered or not clustered at all. As a result, it is not possible to compute the Silhouette score for the clusters obtained from NIS_a. DBSCAN is a widely used clustering algorithm. If the NIS result is not compatible with it, that is a disadvantage of NIS.

Table 6.1 Silhouette Score Table

	original	**NCE 1**	**NCE 2**	**NIS_a**	**NIS_b**
0.0001, 2	0.798	0.566	0.810	—	0.683
0.01, 3	0.639	0.421	0.822	—	0.597
0.001, 3	0.810	0.652	0.859	—	0.663
0.01, 5	0.639	0.464	0.847	—	0.539
0.001, 5	0.639	0.449	0.830	—	0.527
0.0005, 3	0.819	0.663	0.847	—	0.703
0.0005, 5	0.845	0.682	0.848	—	0.736

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets.

The results indicate that NCE 2 achieves the highest Silhouette score among all tested methods, suggesting that it produces the most well-defined and separated clusters. Additionally, the performance of NCE 2 exceeds that of the clustering based on the original, unmodified dataset.

6.1.2 Reduction Rate Results

Table 6.2 displays the reduction rate of the different clustering algorithms with specific clustering arguments. However, it is important to note that the reduction rate is not applicable to the original data set, as it includes the entire data set, and therefore has a 100% reduction rate. The reduction rate measures the percentage of instances that are kept from the data set during the instance selection process.

Table 6.2 Reduction Rate Table

	original	**NCE 1**	**NCE 2**	**NIS_a**	**NIS_b**
0.0001, 2	—	1.0%	1.9%	1.9%	1.9%
0.01, 3	—	3.0%	6.8%	6.8%	6.8%
0.001, 3	—	2.1%	4.8%	4.8%	4.8%
0.01, 5	—	1.8%	5.7%	5.7%	5.7%
0.001, 5	—	1.8%	5.8%	5.8%	5.8%
0.0005, 3	—	1.7%	3.7%	3.7%	3.7%
0.0005, 5	—	1.0%	2.9%	2.9%	2.9%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets.

The results of the reduction rate demonstrate that each algorithm is capable of effectively reducing the size of the dataset as intended, achieving the desired level of data reduction.

6.1.3 Purity Test Results

Table 6.3 displays the purity test results of the different clustering algorithms with specific clustering arguments. However, it is important to note that the Purity Test is not applicable to the result of NIS_a. This is because the NIS algorithm does not cluster the data properly using density-based clustering, and only a small percentage of the data set (less than one percent) is either clustered or not clustered at all. As a result, it is not possible to compute the Purity Test for the clusters obtained from NIS _a. DBSCAN is a widely used clustering algorithm. If the NIS result is not compatible with it, that is a disadvantage of NIS.

Table 6.3 Purity Rate Table

	original	**NCE 1**	**NCE 2**	**NIS_a**	**NIS_b**
0.0001, 2	67%	65%	62%	—	62%
0.01, 3	62%	61%	59%	—	62%
0.001, 3	65%	66%	62%	—	65%
0.01, 5	62%	62%	60%	—	63%
0.001, 5	62%	63%	60%	—	63%
0.0005, 3	67%	66%	64%	—	66%
0.0005, 5	67%	67%	65%	—	66%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets.

The purity test results indicate that, on average, the clusters obtained by the original dataset have the highest purity rate. However, NCE 1 and NIS_b also achieve relatively high purity rates, while NCE 2 yields the lowest purity rate among the tested methods.

6.1.4 Execution Time

Table 6.4 displays the execution time of the different instance selection algorithms with specific clustering arguments. The execution time is measured in seconds, denoted by ”s” in the table.

Table 6.4 Execution Time Table

	NCE 1	**NCE 2**	**NIS_a**	**NIS_b**
0.0001, 2	651s	1,166s	174s	101s
0.01, 3	29,837s	48,940s	229s	108s
0.001, 3	9,498s	17,434s	198s	104s
0.01, 5	18,926s	32,926s	213s	107s
0.001, 5	7,128s	12,722s	231s	106s
0.0005, 3	4,586s	8,543s	205s	103s
0.0005, 5	2,880s	6,832s	180s	104s

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets.

The execution time table reveals that both NCE 1 and NCE 2 require more time to execute compared to the NIS algorithm. Additionally, NIS_a exhibits a longer execution time than NIS_b, as the input is a larger volume of data.

6.2 K-Nearest Neighbor Classification

This section provides details on the setup and outcomes of the k-nearest neighbor (KNN) classification.

6.2.1 Set-Up

The following subsections describe the data used for all KNN classifications, including information on the reduction rate and execution time.

6.2.1.1 Stratification of NCE

The KNN classifier results of the NCE algorithm are obtained by stratifying the EMBER training data set. To achieve this, the NCE algorithm is performed on each training data set of EMBER, and the results are combined.

The execution times of stratification are represented in Table 6.5.

Table 6.5 Execution Time of Stratification Table

	NCE 1	**NCE 2**
0.0001, 2	9,504s	19,113s
0.01, 3	47,365s	71,625s
0.001, 3	14,819s	26,972s
0.01, 5	34,857s	54,555s
0.001, 5	18,379s	29,750s
0.0005, 3	12,661s	25,301s
0.0005, 5	4,952s	10,026s

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

Table 6.5 demonstrates that NCE 2 exhibits a slower execution time compared to NCE 1.

The reduction rates of stratification are represented in Table 6.6.

Table 6.6 Reduction Rate of Stratification Table

	NCE 1	**NCE 2**
0.0001, 2	1.0%	1.9%
0.01, 3	2.5%	5.8%
0.001, 3	1.8%	4.2%
0.01, 5	1.5%	4.9%
0.001, 5	1.0%	3.1%
0.0005, 3	1.3%	3.0%
0.0005, 5	0.6%	2.3%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

Table 6.6 indicates that NCE 1 achieves a higher reduction rate compared to NCE 2.

6.2.1.2 Preparation of NIS

Table 6.7 represents the reduction rate and execution time of the NIS algorithm applied to the entire EMBER training data set. In the KNN classification test, clustering is not a requirement. Therefore, the NIS algorithm is used in its raw form without any clustering. The algorithm takes the entire preprocessed EMBER data set as input and performs instance selection based on the feature vector of the data.

Table 6.7 Execution Time by Reduction Rate of NIS Table

	NIS
0.6	253s
1.0	250s
1.3	254s
1.5	256s
1.8	259s
1.9	255s
2.3	258s
2.5	260s
3.0	264s
3.1	263s
4.2	263s
4.9	262s
5.8	265s

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets.

The execution time by reduction rate table demonstrates that the NIS algorithm is capable of achieving the same reduction rate as NCE while executing in less time. In fact, the NIS algorithm achieves the same reduction rate in a considerably shorter amount of time compared to NCE.

6.2.2 K-Nearest Neighbor Classification Results

Each section below represents the KNN classification results of the NCE algorithm, the NIS algorithm, and the original unreduced EMBER training data set for different values of K.

6.2.2.1 KNN Classification Results of K = 5

The KNN classification results for the entire EMBER training set with a value of K equal to 5 are presented in Table 6.8.

Table 6.8 KNN Classification Results of Original Data Set for K = 5

95.37%

The KNN classification results for NCE 1 and NCE 2 with a value of K equal to 5 are presented in Table 6.9, along with the corresponding clustering parameters.

Table 6.9 KNN Classification Results of NCE for K = 5 Table

	NCE 1	**NCE 2**
0.0001, 2	76.02%	82.54%
0.01, 3	83.6%	79.13%
0.001, 3	80.93%	85.9%
0.01, 5	74.71%	81.82%
0.001, 5	75.53%	81.79%
0.0005, 3	79.23%	84.36%
0.0005, 5	76.12%	79.72%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

On average, the KNN classification results presented in Table 6.9 indicate that the NCE 2 algorithm outperforms NCE 1.

The KNN classification results for NIS with a value of K equal to 5 are presented in Table 6.10, along with the corresponding reduction rates.

Table 6.10 KNN Classification Results of NIS for K = 5 Table

	NIS
0.6	73.97%
1.0	74.35%
1.3	73.05%
1.5	71.67%
1.8	70.61%
1.9	75.6%
2.3	71.33%
2.5	75.9%
3.0	74.93%
3.1	76.62%
4.2	76.88%
4.9	76.75%
5.8	77.41%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

The results of the NIS algorithm will be discussed in the following tables.

The KNN classification results for NIS and NCE 1, both with a value of K equal to 5, are presented in Table 6.11 along with the corresponding reduction rates. It is important to note that there are two rows for a 1.0% reduction rate because there are two NCE 1 with a 1.0% reduction rate.

Table 6.11 KNN Classification Results of NIS and NCE 1 for K = 5 Table

	NIS	NCE 1
0.6	73.97%	76.12%
1.0	74.35%	76.02%
1.0	—	75.52%
1.3	73.05%	79.23%
1.5	71.67%	74.41%
1.8	70.61%	80.93%
2.5	75.9%	83.6%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.11, NCE 1 achieves better performance than the NIS algorithm.

The KNN classification results for NIS and NCE 2, both with a value of K equal to 5, are presented in Table 6.12 along with the corresponding reduction rates.

Table 6.12 KNN Classification Results of NIS and NCE 2 for K = 5 Table

	NIS	NCE 2
1.9	75.6%	82.54%
2.3	71.33%	79.72%
3.0	74.93%	84.36%
3.1	76.62%	81.79%
4.2	76.88%	85.9%
4.9	76.75%	81.82%
5.8	77.41%	79.13%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.12, NCE 2 achieves better performance than the NIS algorithm.

6.2.2.2 KNN Classification Results of K = 9

The KNN classification results for the entire EMBER training set with a value of K equal to 9 are presented in Table 6.13.

Table 6.13 KNN Classification Results of Original Data Set for K = 9

95.15%

The KNN classification results for NCE 1 and NCE 2 with a value of K equal to 9 are presented in Table 6.14, along with the corresponding clustering parameters.

Table 6.14 KNN Classification Results of NCE for K = 9 Table

	NCE 1	NCE 2
0.0001, 2	72.45%	81.49%
0.01, 3	83.22%	78.56%
0.001, 3	79.02%	85.13%
0.01, 5	72.89%	81.8%
0.001, 5	74.52%	81.79%
0.0005, 3	77.65%	84.36%
0.0005, 5	73.92%	79.7%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

On average, the KNN classification results presented in Table 6.14 indicate that the NCE 2 algorithm outperforms NCE 1.

The KNN classification results for NIS with a value of K equal to 9 are presented in Table 6.15, along with the corresponding reduction rates.

Table 6.15 KNN Classification Results of NIS for K = 9 Table

	NIS
0.6	72.44%
1.0	75.1%
1.3	74.49%
1.5	72.73%
1.8	73.79%
1.9	73.97%
2.3	73.73%
2.5	75.14%
3.0	74.46%
3.1	74.96%
4.2	77.23%
4.9	76.28%
5.8	77.33%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

The results of the NIS algorithm will be discussed in the following tables.

The KNN classification results for NIS and NCE 1, both with a value of K equal to 9, are presented in Table 6.16 along with the corresponding reduction rates. It is important to note that there are two rows for a 1.0% reduction rate because there are two NCE 1 with a 1.0% reduction rate.

Table 6.16 KNN Classification Results of NIS and NCE 1 for K = 9 Table

	NIS	NCE 1
0.6	72.44%	73.92%
1.0	75.1%	72.45%
1.0	—	74.52%
1.3	74.49%	77.65%
1.5	72.73%	72.89%
1.8	73.79%	79.02%
2.5	75.14%	83.22%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.16, NCE 1 achieves better performance than the NIS algorithm on average.

The KNN classification results for NIS and NCE 2, both with a value of K equal to 9, are presented in Table 6.17 along with the corresponding reduction rates.

Table 6.17 KNN Classification Results of NIS and NCE 2 for K = 9 Table

	NIS	NCE 2
1.9	73.97%	81.49%
2.3	73.73%	79.7%
3.0	74.46%	84.36%
3.1	74.96%	81.79%
4.2	77.23%	85.13%
4.9	76.28%	81.8%
5.8	77.33%	78.56%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.17, NCE 2 achieves better performance than the NIS algorithm.

6.2.2.3 KNN Classification Results of K = 13

The KNN classification results for the entire EMBER training set with a value of K equal to 13 are presented in Table 6.18.

Table 6.18 KNN Classification Results of Original Data Set for K = 13

94.87%

The KNN classification results for NCE 1 and NCE 2 with a value of K equal to 13 are presented in Table 6.19, along with the corresponding clustering parameters.

Table 6.19 KNN Classification Results of NCE for K = 13 Table

	NCE 1	NCE 2
0.0001, 2	69.68%	80.11%
0.01, 3	81.93%	77.43%
0.001, 3	78.7%	84.8%
0.01, 5	71.24%	81.97%
0.001, 5	73.35%	81.75%
0.0005, 3	76.86%	83.86%
0.0005, 5	72.44%	79.99%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

On average, the KNN classification results presented in Table 6.19 indicate that the NCE 2 algorithm outperforms NCE 1.

The KNN classification results for NIS with a value of K equal to 13 are presented in Table 6.20, along with the corresponding reduction rates.

Table 6.20 KNN Classification Results of NIS for K = 13 Table

	NIS
0.6	72.04%
1.0	74.37%
1.3	74.3%
1.5	72.25%
1.8	73.81%
1.9	73.71%
2.3	74.84%
2.5	74.06%
3.0	75.8%
3.1	73.62%
4.2	76.81%
4.9	76.23%
5.8	76.39%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

The results of the NIS algorithm will be discussed in the following tables.

The KNN classification results for NIS and NCE 1, both with a value of K equal to 13, are presented in Table 6.21 along with the corresponding reduction rates. It is important to note that there are two rows for a 1.0% reduction rate because there are two NCE 1 with a 1.0% reduction rate.

Table 6.21 KNN Classification Results of NIS and NCE 1 for K = 13 Table

	NIS	NCE 1
0.6	72.04%	72.44%
1.0	74.37%	69.68%
1.0	—	73.35%
1.3	74.3%	76.86%
1.5	72.25%	71.24%
1.8	73.81%	78.7%
2.5	74.06%	81.93%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.21, NCE 1 achieves better performance than the NIS algorithm on average.

The KNN classification results for NIS and NCE 2, both with a value of K equal to 13, are presented in Table 6.22 along with the corresponding reduction rates.

Table 6.22 KNN Classification Results of NIS and NCE 2 for K = 13 Table

	NIS	NCE 2
1.9	73.71%	80.11%
2.3	74.84%	77.43%
3.0	75.8%	83.86%
3.1	73.62%	81.75%
4.2	76.81%	84.8%
4.9	76.23%	81.97%
5.8	76.39%	77.43%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.22, NCE 2 achieves better performance than the NIS algorithm.

6.2.2.4 KNN Classification Results of K = 17

The KNN classification results for the entire EMBER training set with a value of K equal to 17 are presented in Table 6.23.

Table 6.23 KNN Classification Results of Original Data Set for K = 17

94.68%

The KNN classification results for NCE 1 and NCE 2 with a value of K equal to 17 are presented in Table 6.24, along with the corresponding clustering parameters.

Table 6.24 KNN Classification Results of NCE for K = 17 Table

	NCE 1	NCE 2
0.0001, 2	67.82%	78.95%
0.01, 3	81.45%	77.25%
0.001, 3	78.57%	84.32%
0.01, 5	70.59%	80.87%
0.001, 5	72.01%	81.21%
0.0005, 3	75.65%	83.62%
0.0005, 5	72.32%	79.37%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are different NCE elimination techniques.

On average, the KNN classification results presented in Table 6.24 indicate that the NCE 2 algorithm outperforms NCE 1.

The KNN classification results for NIS with a value of K equal to 17 are presented in Table 6.25, along with the corresponding reduction rates.

Table 6.25 KNN Classification Results of NIS for K = 17 Table

	NIS
0.6	72.66%
1.0	73.9%
1.3	80.02%
1.5	72.05%
1.8	73.66%
1.9	73.15%
2.3	72.74%
2.5	74.75%
3.0	75.41%
3.1	74.67%
4.2	75.88%
4.9	76.09%
5.8	76.6%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

The results of the NIS algorithm will be discussed in the following tables.

The KNN classification results for NIS and NCE 1, both with a value of K equal to 17, are presented in Table 6.26 along with the corresponding reduction rates. It is important to note that there are two rows for a 1.0% reduction rate because there are two NCE 1 with a 1.0% reduction rate.

Table 6.26 KNN Classification Results of NIS and NCE 1 for K = 17 Table

	NIS	NCE 1
0.6	72.66%	72.32%
1.0	73.9%	67.82%
1.0	—	72.01%
1.3	80.02%	75.65%
1.5	72.05%	70.59%
1.8	73.66%	78.57%
2.5	74.75%	81.45%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.26, NIS achieves better performance than the NCE 1 algorithm on average.

The KNN classification results for NIS and NCE 2, both with a value of K equal to 17, are presented in Table 6.27 along with the corresponding reduction rates.

Table 6.27 KNN Classification Results of NIS and NCE 2 for K = 17 Table

	NIS	**NCE 2**
1.9	73.15%	78.95%
2.3	72.74%	79.37%
3.0	75.41%	83.62%
3.1	74.67%	81.21%
4.2	75.88%	84.32%
4.9	76.09%	80.87%
5.8	76.6%	77.25%

The rows of the table are arguments of clustering, the left-hand side of the coma is the maximum distance between instances in a cluster and the right-hand side of the coma is the minimum number of instances in a cluster. The columns of the tables are the data sets

Based on the KNN classification results presented in Table 6.11, NCE 2 achieves better performance than the NIS algorithm.

Chapter 7
The Conclusion

This study aimed to provide a comparative analysis of two unsupervised instance selection algorithms, the Nearest Cluster Enemy (NCE) algorithm, and the Nimble Instance Selection (NIS) algorithm. The primary objective was to propose an unsupervised instance selection algorithm and offer valuable insights into the strengths and weaknesses of both algorithms. The ultimate goal was to help researchers select the most appropriate algorithm for their specific application.

The results of this study showed that the NIS algorithm outperformed the NCE algorithm in terms of computational efficiency, as it had lower execution times for both techniques. Moreover, the purity of clustered results of NCE 1 and NIS was better than that of NCE 2 results, indicating that NCE 2 may not be the optimal choice.

However, the Silhouette scores revealed that the quality of clusters created by the NCE 2 algorithm was better than those created by NCE 1 and NIS, suggesting that NCE 2 may be more suitable for certain applications. Additionally, the KNN classification results of NCE 2 were consistently better than those of NIS with the same reduction rate. On the other hand, NCE 1 results were better than NIS on average, although the difference was relatively small compared to the difference between NIS and NCE 2.

It is worth noting that the NIS algorithm's results are not compatible with density-based clustering, which is a popular technique for clustering unlabeled data. This limitation may make it unsuitable for certain applications. However, it is essential to consider the computational efficiency, clustering quality, and compatibility with other techniques when choosing an algorithm for a specific application.

In conclusion, this study provided valuable insights into the strengths and weaknesses of two unsupervised instance selection algorithms and their elimination techniques. By highlighting the performance of each algorithm, researchers can use these insights to choose the most appropriate algorithm for their specific application, depending on the factors mentioned above. Ultimately, this study contributes to the development of unsupervised instance selection algorithms and their applications.

Bibliography

1. JUREČEK, Martin; JUREČKOVÁ, Olha. Parallel Instance Filtering for Malware Detection. In: *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2022, pp. 13–20. Available from DOI: 10.1109/SEAA56994.2022.00012.
2. HART, P. The condensed nearest neighbor rule (Corresp.) *IEEE Transactions on Information Theory*. 1968, vol. 14, no. 3, pp. 515–516. Available from DOI: 10.1109/TIT.1968.1054155.
3. WILSON, Dennis L. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics*. 1972, vol. SMC-2, no. 3, pp. 408–421. Available from DOI: 10.1109/TSMC.1972.4309137.
4. BRIGHTON, Henry; MELLISH, Chris. Advances in Instance Selection for Instance-Based Learning Algorithms. *Data Mining and Knowledge Discovery*. 2002, vol. 6, no. 2, pp. 153–172. Available from DOI: 10.1023/A:1014043630878.
5. WILSON, D. Randall; MARTINEZ, Tony R. Reduction Techniques for Instance-Based Learning Algorithms. *Machine Learning*. 2000, vol. 38, no. 3, pp. 257–286. Available from DOI: 10.1023/A:1007626913721.
6. AV-TEST GMBH. *AV-TEST Statistics* [https://www.av-test.org/en/statistics/malware/]. Accessed on May 1, 2023.
7. MEFEZ. *Unlabeled Ins Selection* [https://github.com/Mefez/Unlabeled_Ins_Selection]. 2021. Accessed: [Insert Date].
8. ANDERSON, Hyrum S; ROTH, Philip; KLIGER, Max; STORLIE, Curtis B. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1–9.
9. DEVELOPERS, scikit-learn. StandardScaler. *scikit-learn: Machine Learning in Python*. 2021. Available also from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.
10. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, 'E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.
11. SCIKIT-LEARN CONTRIBUTORS. *sklearn.decomposition.PCA* [https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html]. Accessed on May 1, 2023.

12. YU, Sheng; ZHOU, Shijie; LIU, Leyuan; YANG, Rui; LUO, Jiaqing. Detecting Malware Variants by Byte Frequency. *JNW*. 2011, vol. 6, pp. 638–645. Available from DOI: 10.4304/jnw.6.4.638-645.

13. TABISH, S. Momina; SHAFIQ, M. Zubair; FAROOQ, Muddassar. Malware Detection using Statistical Analysis of Byte-Level File Content. In: *2013 5th International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2013, pp. 139–144.

14. AYDIN, Fatih. Unsupervised instance selection via conjectural hyperrectangles. *Neural Computing and Applications*. 2023, vol. 35, no. 7, pp. 5335–5349. ISSN 1433-3058. Available from DOI: 10.1007/s00521-022-07974-z.

15. SCIKIT-LEARN CONTRIBUTORS. *sklearn.neighbors.KNeighborsClassifier* [https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html]. Accessed on May 1, 2023.

Appendix A
Acronyms

PIF = Parallel Instance Filtering
AI = Artificial Intelligence
ML = Machine Learning
CNN = Condensed Nearest Neighbor
ENN = Edited Nearest Neighbor
ICF = Iterative Case Filtering
DROP = Discriminative Random Over-Sampling Projection
NIS = Nimble Instance Selection
NCE = Nearest Cluster Enemy
NE = Nearest Enemy
PCA = Principal component analysis
EMBER = Endgame Malware BEnchmark for Research
ELF = Executable and Linkable Format
PE = Portable Executable
Mach-O = Mach Object
DBSCAN = Density-Based Spatial Clustering of Applications with Noise
K-NN = K-Nearest Neighbors