RansomFoRRT: Better Features for Robust Ransomware Threat Detection

by

Rajbir Bhattacharjee

This thesis has been submitted in partial fulfillment for the degree of Master of Science in Artificial Intelligence

in the
Faculty of Engineering and Science
Department of Computer Science

May 2022

Declaration of Authorship

This report, RansomFoRRT: Better Features for Robust Ransomware Threat Detection, is submitted in partial fulfillment of the requirements of Master of Science in Artificial Intelligence at Munster Technological University Cork. I, Rajbir Bhattacharjee, declare that this thesis titled, RansomFoRRT: Better Features for Robust Ransomware Threat Detection and the work represents substantially the result of my own work except where explicitly indicated in the text. This report may be freely copied and distributed provided the source is explicitly acknowledged. I confirm that:

- This work was done wholly or mainly while in candidature Master of Science in Artificial Intelligence at Munster Technological University Cork.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Munster Technological University Cork or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

MUNSTER TECHNOLOGICAL UNIVERSITY CORK

Abstract

Faculty of Engineering and Science
Department of Computer Science

Master of Science

by Rajbir Bhattacharjee

In recent years, there has been a rise in ransomware attacks, with losses amounting to the tens of billions and resulting in public crises. Fortunately, ransomware detection is an active area of research. However, there are still some challenges lacking adequate solutions.

The first challenge is ransomware that bypass detection by using adversarial techniques or by tweaking their encryption algorithm. Most commercial ransomware detection solutions fall short as they only use a small set of statistical measures and this makes bypassing them easy.

The second challenge is avoiding false positives in detection. False positives occur because it is difficult to distinguish between encrypted files, password protected files, compressed files, and certain high entropy file formats like webp. Files that were not included while training the model may also cause false positives.

In this work, I take a combination of autocorrelation, chi-square statistics, Monte-Carlo estimation of π, central and standardized moments, generalized entropies, and Fourier power spectral density and demonstrate that this combination is effective in overcoming both these challenges. This results in an overall maximum improvement of 0.225 in the F1 score. For webp files, office files, password protected files, and archives, the maximum improvements in F1 score are 0.173, 0.247, 0.238, and 0.047 respectively. For files that were not included when training the model, the overall maximum improvement is 0.162.

To the best of my knowledge, this is also the first work to systemically study the use of Fourier power spectral density in detecting encrypted content in files.

Acknowledgements

In the months during which I worked on this project, and the months leading up to it, many people have encouraged me and helped me shape my ideas and the nature of my inquiries.

Foremost, I owe much gratitude to my project supervisor, Alex Vakaloudis, without whose help and guidance, it would not have been possible to complete this project. Alex was always kind to me and his encouragement and direction were key to the development of the ideas that were explored here.

It was in Christian Beder's class where I re-learned Fourier analysis and where the first seeds of this idea were sown. When I was filled with doubts and a fear of failure, Christian encouraged me to pursue what I was passionate about. Thank you, Christian.

For teaching me the fundamentals of machine learning, and for always being patient with my many questions, I would like to thank Ted Scully.

Steve Brunton's lectures on Fourier analysis and Christof Paar's lectures on cryptography helped me understand many nuances. Justin Zeltzer's YouTube channel was indispensable in understanding many statistical concepts quickly. I am obliged to them and the many other educators who share their knowledge on the internet for free.

I would like to thank Tanya Raghuvanshi for the many stimulating discussions which encouraged me to learn artificial intelligence. And I owe my sincerest gratitude to the late Ashoke Bijoy Chakraborty for teaching me calculus twenty-two years ago. It is only because of the foundations he laid then that I ever dreamed of studying artificial intelligence. Over the years, I have benefited from my association with Chandanathil Pappachan Geevan, and his review comments helped me refine my ideas.

I am grateful to Rachana Bhattacharjee for proof reading and editing my thesis drafts over the course of its development.

Finally, I owe many thanks to all those who have supported me elsewhere so that I may focus my time on this research: Baishali Chetia, without whose encouragement and understanding, I would not have joined college again; and my aunt, Chandra Bhatacharjee, and my father, Jahar Bhattacharjee, who have always believed in me and have always encouraged me in everything that I did.

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
API	**A**pplication **P**rogramming **I**nterface
OS	**O**perating **S**ystem
KNN	**k**-**N**earest **N**eighbours
PE	**P**ortable **E**executable
BFD	**B**yte **F**requency **D**istribution
BFC	**B**yte **F**requency **C**ross-correlation
DFT	**D**iscrete **F**ourier **T**ransform
FFT	**F**ast **F**ourier **T**ransform
CNN	**C**onvolutional **N**eural **N**etwork
ROC	**R**eceiver **O**perating **C**haracteristic
AUC	**A**rea **U**nder the ROC **C**urve
I/O	**I**nput / **O**utput
ReLU	**Re**ctified **L**inear **U**nit
RAAS	**R**ansomware **A**s **A** **S**ervice
AES	**A**dvanced **E**ncryption **S**tandard
3DES	Triple **D**ata **E**ncryption **S**tandard
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
NPTEL	**N**ational **P**rogramme on **T**echnology **E**nhanced **L**earning

For Ma, Mashimoni, Moonmashi and Gultu

Chapter 1

Introduction

Ransomware is malware that is designed to extract a ransom from users. The most common class of ransomware prevents users from accessing their computers or their files. A second class of ransomware threatens to release confidential data in the public sphere unless a ransom is paid.

The estimated losses from ransomware attacks vary. The Federal Bureau of Investigation estimated that in 2020 that ransomware causes a loss of 29 billion dollars [2]. Apart from the financial loss, ransomware can also cause public emergencies. Ireland's Health Service Executive was the target of a Conti ransomware attack that caused delays in diagnostics, the reporting of results, prescription of medication and treatment of patients [3–7].

McIntosh et al. [1] divide ransomware into the following categories:

1. **Hoaxware or scareware.** These are unwanted programs that display a scare message with the hope of extracting a ransom. They cause no other damage and are easy to remove.

2. **Screen lockers.** These are unwanted programs that lock the whole operating system and restrict user access to data. This type of malware does not actually encrypt any data and is easy to remediate.

3. **Crypto-ransomware.** These encrypt user data and only decrypt them when a ransom is paid.

4. **File-less ransomware.** These are not distributed as executable files. They use executable files and utilities already present in the user's computer in nefarious ways to extract ransoms using any of the first three techniques.

5. **VM-based ransomware.** These are distributed as complete virtual machines. This is to avoid detection by scanning engines which look for typical access patterns and the signatures of existing ransomware.

6. **Ransomware with data ex-filtration.** This type of ransomware ex-filtrates confidential user data and threatens to release them in the public domain if a ransom is not paid.

CrowdStrike Inc. [8] uses a similar classification but adds a category for *ransomware as a service* or *RAAS*. This refers to malware that is developed and hosted by someone else in return for a cut of the ransom.

Mohammad [9] uses a simpler classification of ransomware into just two categories: crypto-ransomware and locker ransomware.

1.1 Motivation

With the rise of cryptocurrencies, accepting payments from victims has become easier. Attackers can now demand ransom in crptocurrencies which are outside the purview of law enforcement. This has been partly responsible for an increase in ransomware attacks.

There has also been an increase in ransomware that uses adversarial techniques to avoid detection [10]. It has been also demonstrated that current detection techniques can be bypassed by trivial methods [10–12].

Fortunately, ransomware defence has been an active area of research in recent years. McIntosh et al. [1] present the number of results in Google Scholar for ransomware by year in Figure 1.1.

While there has been an increased interest in ransomware defence in academia and industry, most work in this field has suffered from some gaps:

1. Existing methods fare poorly against adversarial techniques [11].

2. The method of evaluation of the performance in existing works is not always clear.

   (a) Most existing studies do not use standard datasets and rely on private datasets.

   (b) These datasets are not adequately described, in terms of diversity of file types, number of files, file sizes, etc.

(c) Many studies use very small datasets which are not representative of file types in the wild.

(d) Many of the existing approaches rely on reading the first few bytes of a file which contain a predictable header to give better accuracy, but ransomware in the wild are known to avoid encrypting the first few bytes [10, 12].

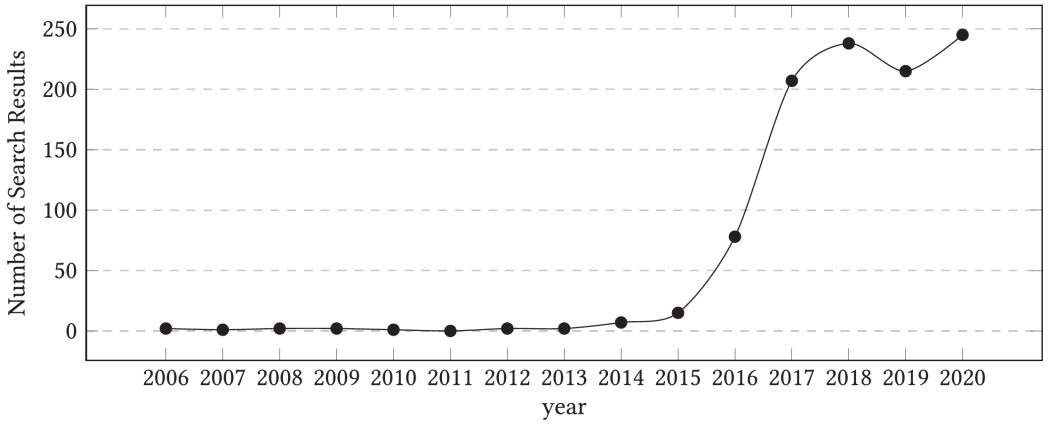

FIGURE 1.1: The Number of articles with titles containing the keyword "Ransomware" per year on Google Scholar, McIntosh et al. [1]

1.2 Contribution

This work further contributes to ransomware defence by identifying encrypted content in user data towards the goal of robust and resilient encryption detection against a variety of file types, encryption schemes, and adversarial attacks. Robust detection of encrypted content can be a critical piece of the puzzle for mitigating ransomware threat, especially in cloud backup services, which can undertake mitigating actions if they detect that a user's files have been compromised by ransomware.

This work takes a data centric approach rather than a model centric approach. The goal of this work is not to design a model that is the best at detecting ransomware; rather, the goal in this project is to find what features are most effective in detecting ransomware even when used with a wide variety of machine learning models.

I study a host of features, and their combinations, many of which have not been studied before, and demonstrate that the combinations of these features result in:

1. Increased accuracy of detection, including the detection of previously unseen file types.
2. More consistent detection across samples.

3. Resilience against adversarial attacks.
4. Better accuracy and fewer false positives when classifying webp files, office files, password proteced files, and archives.

These features include autocorrelation, various central and standardized moments, chi-square satistics, various generalized entropies, Monte-Carlo simulation of π, and Fourier power spectral density. Some of these statistical features have not been evaluated for their effectiveness in ransomware detection in a systematic manner in past studies. I am not aware of any previous study that has evaluated the use of Fourier power spectral density in ransomware detection by identifying encryption in files.

I further demonstrate that the most accurate and robust detection can be obtained by combining these features.

I also demonstrate that these features are effective in detecting ransomware when the file has only been partially encrypted, or has been further processed after encryption to reduce entropy.

I show that these features perform superiorly to existing methods across a variety of machine learning models, which can be either simple or complex.

In Section 5.6, I show that these features are able to perform well when presented with new file types that were not used in training. I further show that Fourier transforms can help distinguish between encrypted, password protected, and compressed files—other features are unable to distinguish between them.

Lastly, I also demonstrate that just a sample of bytes from files can be used for the resource-efficient detection of encrypted content.

1.3 Potential Uses

The potential uses of this work fall under two categories:

1. It can be used to detect ransomware in end-point computing devices like laptops, tablets, etc.
2. It can be used in cloud backup system servers to detect ransomware attacks on client machines, and take appropriate action.

Cloud backup systems have a legitimate interest in detecting attacks in client machines as they can take remedial action if they suspect such an attack. One such remedial action

would be to create a copy of a file instead of overwriting it, if there is a suspicion that it was modified by ransomware. Cloud backup services could provide anti-ransomware services by creating a copy of a file every time it is modified, but this approach would be wasteful of storage. If they have a way of knowing whether something was legitimately password protected or encrypted by ransomware, they can make a more intelligent decision on whether to overwrite the backup or create a copy. For a cloud backup service, this would translate to optimized storage and lower costs. While there are many ways in which ransomware can be detected on the end point (e.g. static and dynamic analyses), there aren't many ways in which cloud backup systems can detect that their client machines have been affected by ransomware attacks. Detecting encryption in backed up files is one of the few ways in which cloud backup services can detect ransomware attacks on their client machines. However, detecting encryption in files is the least studied method of ransomware detection, and hence, is the focus of this work.

1.4 Structure of this Document

I perform a review of the existing literature in Chapter 2 and identify the gaps in section 2.8. In Chapter 3, I illustrate the need for better features by demonstrating a trivial attack on a trained neural network that detects files encrypted by ransomware. I describe the high-level design of my approach in Chapter 4. I list the project objectives and requirements and follow it with a discussion on my choice of features and the theoretical considerations. Chapter 5 describes my implementation and evaluation of results. The work was done in a series of agile sprints. I begin with an overview of sprints and a description of the datasets used, and then describe each sprint. I conclude this report in Chapter 6 with a discussion on my results and directions for future work.

Chapter 2

Literature Review

2.1 Overview

In Chapter 1, I looked at why there is a need for further research in the domain of ransomware detection. Here I present a comprehensive literature review to understand the current approaches to ransomware detection, and where there is scope of improvement.

I begin by looking at the different ways ransomware detection techniques have been classified in the literature. I then conduct a thematic review of different ransomware detection techniques:

1. Static binary analysis
2. Dynamic analysis
3. Detecting encrypted content in user data
4. File fragment type identification
5. Frequency domain analysis

Static and dynamic analysis methods have received a lot of attention in the literature, but these are not a focus of this work and therefore, I do not perform an extensive review. However, I review some selected literature on these topics for a holistic view of ransomware detection techniques. File fragment type identification is another task that overlaps with ransomware detection, and hence I review the literature that explores this task. Frequency domain analysis has been applied in dynamic analysis techniques to detect ransomware, and although this is not the focus of this work, I do a limited review of such techniques because I employ frequency domain analysis to detect encryption in users data. Finally, I present what I think are gaps in the existing literature.

2.2 Classification of Ransomware Detection Techniques

Ransomware detection techniques can be classified in various ways.

Kok et al. [13] use the following classification ransomware detection techniques:

1. **Machine learning based.** Machine learning algorithms can be used to perform static or dynamic analyses of executable binaries to detect ransomware.
2. **Honeypot based.** Decoy canary files are set up in the monitored system. If these files are encrypted, this is an indication of ransomware activity.
3. **Statistics based.** Statistical measurements like entropy can be used to detect ransomware. Such methods can be used on both executable binary files and user data.

Davies et al. [14] suggest a similar classification:

1. Rule-based
2. Statistical analysis
3. Behavioural analysis
4. Machine learning

McIntosh et al. [1] make a more nuanced classification of detection techniques:

1. Hardware level techniques. Patterns of branches, cache-misses, etc. can be used to recognize patterns that can help ransomware detection.
2. Kernel mode techniques. Kernel mode drivers can be used to profile interaction with kernel APIs to detect ransomware via the recognition of patterns and signatures.
3. User mode techniques. These detection techniques run in the user space and are easier to implement. McIntosh et al. further sub-classify these techniques into
   (a) API, Opcode or language run-time analysis. This involves static analysis of executable binaries to detect API calls or opcode patterns that are associated with ransomware.
   (b) Detection of OS compromise, like modification of system registries
   (c) Encryption activity detection
   (d) File system activity analysis

(e) Network traffic analysis. These techniques assume communication with a command and control centre, and aim to detect ransomware activity using analysis of network traffic.

According to Moussaileb et al. [15], every ransomware goes through a few stages in its attack. Detection and prevention mechanisms can be classified according to which stage it targets. They provide a classification of mitigation techniques at each of these stages. The stages and their respective mitigation techniques are:

1. Delivery. How did the ransomware get to the system?
   Mitigation:
   (a) User security awareness
   (b) Data backup
   (c) Access control lists
   (d) Volume shadow copy and backups
   (e) Signature based detection
2. Environment preparation. This might involve changes to the system to avoid detection or ensure the malware persists.
   Mitigation:
   (a) Monitor API calls and windows events
3. Encryption, file and network activity
   Mitigation:
   (a) Network analysis
   (b) Network honeypots
   (c) System honeypots
   (d) Moving target defence
   (e) File monitoring
   (f) Hardware performance counters with ML models
   (g) Backup of keys
4. Ransom payment
   Mitigation:
   (a) Bitcoin payment tracking

In this review, I use a simpler classification of ransomware detection techniques, namely:

(T1) Static analysis of executable binaries

(T2) Dynamic analysis
(T3) Detecting encrypted content in user data

In addition, in this report I give special attention to the use of frequency domain analysis in dynamic analysis techniques. I also look at the task of file fragment categorization as it overlaps with this work.

In the literature I reviewed, the majority of the techniques could be classified under either (T1) or (T2). Comparatively, very few papers described techniques that could be classified under (T3).

2.3 Static analysis of executable binaries

One way to detect ransomware is to analyze binaries and look for signatures. This operation is similar to how traditional antivirus solutions work. The binary is scanned, and whenever a previously known pattern is found, it can be classified as a ransomware. Other variations may apply machine learning where there are no pre-defined patterns; the model learns how to recognize ransomware executable binaries from training data.

There is a large body of work that relies on executable binary analysis to detect ransomware. It is not possible to do a complete literature review of all such techniques. Furthermore, these techniques are not the focus of this report. However, I discuss a few of these techniques to give an idea of how they can be used.

Guo et al. [16] detect ransomware by analyzing executable binaries. They extract features from the executable binary and then train a classifier on 20,000 samples to recognize ransomware. For the feature set, they use entropy and wavelet decomposition grouped into bags of words.

Zhang et al. [17] use N-grams of opcode sequences in ransomware binaries along with a secondary classifier to detect ransomware. For the secondary classifier, they compare the performance of the KNN classifier, Naive Bayes classifier, and Gradient Boosted Decision Tree classifier.

Manavi and Hamzeh [18] extract the PE header [19] of executable binaries, and convert that to a grayscale image. They then pass it through a convolutional neural network and use it to detect ransomware. They train and evaluate this model over a dataset of 2000 executable binaries and report an F1 score of 0.90.

Nataraj et al. [20] use a static analysis method to detect ransomware. Their method involves conversion of executable binary files to grayscale images. Gist features that

describe the image texture are calculated. These are then used to classify whether a given executable file is a ransomware or not.

2.3.1 Evaluation of Methods that Perform Static Binary Analysis

A static analysis of executable binaries is the most direct approach to ransomware detection. It can be effective in detecting existing ransomware in the wild. This, essentially, has all the advantages and drawbacks of traditional antivirus solutions. The reliance on patterns in the executable binaries implies that new ransomware may not be detected.

In many cases, ransomware may not be distributed as executable binaries, and may *live off the land*. In other cases, ransomware executable binaries may mutate very quickly. They may also be coded in non-native languages to avoid detection; malware written in GoLang is gaining popularity [21]. GoLang binaries can carry a lot of boilerplate opcodes, the actual ransomware logic may only be a small part of the entire executable and represented in GoLang's internal format. Another disadvantage of this method is that a model trained on one operating system and architecture will not work in another.

Static analysis is also not possible in backup systems. In such systems, a direct detection of encryption in files is desirable. If such a detection is made, the backup system can create a backup copy of the file before overwriting the contents with the encrypted version.

2.4 Dynamic Analysis

Dynamic analysis collects the sequence of actions performed by a ransomware and uses this to detect if something is ransomware or not. The sequence of actions can be quite varied.

Dynamic analysis is not the focus of this work. However, I will discuss a few approaches to give an idea of dynamic analysis.

Alam et al. [22] use the sequence of hardware events (like branch instructions, cache references, etc.). They use two auto-encoders for classification, and also use fast Fourier transforms to convert the series of hardware operations to the frequency domain.

Ayub et al. [23] collect file system activity operations, group them by process id. They then classify them as ransomware or non-ransomware by use of a fully-connected deep neural network.

Cusack et al. [24] use network flow information with a random forest classifier to detect ransomware.

2.4.1 Evaluation of Methods that Perform Dynamic Analysis

Dynamic analysis has many advantages over static analysis. Static analysis can only decide whether something is a ransomware or not by looking at the op-codes. This leaves room for use of obfuscation to avoid detection. Many malware binaries may have instructions which are encrypted or packed. Other malware may rely completely on *living off the land* to avoid detection. These limitations are addressed by dynamic analysis.

However, dynamic analysis is not completely immune to detection avoidance. Ransomware may use adversarial techniques to avoid detection. Where API call information is used to detect ransomware, detection may be avoided by calling system calls directly. Some ransomware in the wild have gone to extreme lengths to avoid detection. Symantec reported that many ransomware binaries are now installing virtualization software on the victim machine and running from within the virtual machines to avoid detection [25].

On Windows and Linux, it is relatively easy to gather data for dynamic analysis, but the same is not true for other operating systems, especially mobile operating systems where the ability of security software to hook into APIs or use kernel extensions is limited. Therefore dynamic analysis is not a practical solution for these systems where access is restricted.

Dynamic analysis may also be resource intensive as correlating different events across time is not linear in terms of time and space complexity.

As with static analysis, dynamic analysis is also not suitable for backup systems.

2.5 Detecting Encrypted Content in User Data

Detecting whether individual files or buffers are encrypted or not can be another way of detecting ransomware. If a program writes encrypted buffers into most of the files it touches, it is very likely that the process is ransomware. In cases where ransomware uses *live off the land* techniques, the process tree may be analyzed for further attribution.

Furthermore, this technique is not limited to user data and can also be used to detect malware. Instructions in many malware binaries are hidden by multiple layers of encryption to make it more difficult for manual or automated analysis to flag them.

These have been the most commonly used techniques in commercial anti-ransomware solutions, but these are the least explored in academic research.

The core idea is that encryption tends to produce bytes that seem uniformly random, and statistical tests can be used to detect that randomness.

Most commercial solutions use just a few statistical measures, and train a secondary classifier based on those features. Commonly used statistical measures are as follows:

- Shannon's entropy
- Chi-square
- Autocorrelation coefficient
- Monte-Carlo simulation of π

Penrose et al. [26] do a review of the most common statistical measurements that can be used to detect randomness in file fragments.

Lee et al. [27] use the following metrics to distinguish between encrypted and non-encrypted files:

1. Most common value estimate
2. Collision test estimate
3. Markov test estimate
4. Compression test estimate
5. Shannon's entropy

They then use a variety of secondary classifiers to classify files as encrypted or non-encrypted. They report an F1 score of 1.0, however they do not shed any light on what their data source is, and how large and diverse it is.

Scaife et al. [28] use Shannon's entropy and sdhash [29] along with a secondary classifier to detect encryption.

Bat-Erdene et al. [30] use entropy analysis to detect packed content in executable binaries. This problem is very similar to detecting ransomware by encryption detection.

Davies et al. [31] use entropy analysis along with the file type to detect encrypted content. Their idea is that instead of using a single entropy cut-off for all file types,

classification can achieve higher accuracy by having different cut-offs for different file types.

Jung and Won [32] use the entropy of the first 100 bytes of a file to detect encrypted content. They use a set of 200 PDF files to run their tests. A disadvantage of their approach is that many common ransomware samples avoid encrypting the first few bytes of a file to avoid detection.

While the above techniques can detect encrypted content for some file types very well, they are not very robust. Pont et al. [33] point out that these techniques are completely unable to distinguish between encrypted and non-encrypted *webp* files. Furthermore, ransomware can alter their behaviour to perform an entropy reduction step to avoid detection. McIntosh et al. [34] demonstrate that reducing entropy after encryption can be trivially performed by base-64 or base-32 encoding. Some ransomware use other methods to reduce entropy. The Lockfile ransomware encrypts every alternate 16 bytes of a file rather than encrypt the complete file. This results in a lower entropy and most commercial anti-ransomware solutions fail to detect it [10].

The ability of ransomware to trivially bypass anti-ransomware solutions by making slight modifications in their algorithm can be countered by using a greater diversity of statistical measures. The idea here is that it is easy to bypass a test that uses only a few statistical measures by careful engineering, but it becomes more difficult as more statistical tests are used. Out of an array of statistical measures, only one of them needs to fire to provide an effective detection.

Hu et al. [35] use a different and more diverse set of sixteen statistical measurements along with a secondary classifier to detect ransomware:

1. Filename extension
2. Reyni entropy ($\alpha \in \{2, 5\}$, extremity and entire file)
3. Shannon entropy (bits and bytes, extremity and entire file)
4. Compression ratio (extremity and entire file)
5. Most common value test estimate (extremity and entire file)
6. Markov estimate (extremity and entire file)
7. Zero value count for file (bits, extremiity and entire file)

Bailluet et al. [36] take the first few bytes of every file, and build a statistical model using Markov chains. They then use variants of Kullback-Leibler divergence to detect variation from the baseline. Their hypothesis is that the first few bytes of every file contain a header that is characteristic to a file type. However, the drawback of this

approach is that it will not work well with many ransomware which avoid encrypting the first few bytes to avoid detection.

Zhao et al. [37] evaluate a battery of 188 statistical tests for randomness applied to encrypted data identification on enwik8, doc, exe, flv, jpeg, mp3 and pdf files. They conclude that the combination of these tests provides the best discrimination between encrypted and non-encrypted files. However, their data-set is extremely small and consists only of seven files.

Some of the studies discussed in this section don't mention size of the dataset used. For the studies that do, Table 2.1 summarizes them. The mean dataset size is 2252 samples and the median is 683 samples.

Year	Author(s)	Reference	Dataset Size
2010	Roussev, V	[29]	7
2011	Zhao et al.	[37]	7
2016	Scaife et al.	[28]	5099
2017	Bat-Erdene et al.	[30]	365
2018	Jung and Won	[32]	200
2019	Lee et al.	[27]	1200
2021	Hsu et al.	[35]	1000
2021	Bailluet et al.	[36]	10134

TABLE 2.1: Sizes of datasets used in selected studies

2.5.1 Evaluation of Methods that Detect Encrypted Content in User Data

One of the biggest advantages of this approach is its suitability for cloud backup systems. In the event of a ransomware attack, the ransomware will encrypt all files resident in the user's system. Some of these files will be synced with cloud, and the original files will be overwritten by their encrypted versions. Thus, the user will not be able to restore the file from the cloud. Encryption detection can help break this attack. If the cloud backup system can detect that a file has been encrypted, it can take a snapshot of the file on sync before overwriting it.

2.6 Frequency Domain Analysis

There has been some work that leverages frequency domain analysis in both static and dynamic analyses to detect ransomware. I review a few of these approaches since frequency domain analysis is a key part of this project.

Guo et al. [16] use a static analysis approach to detect ransomware executable binaries. They apply wavelet transforms on blocks of bytes and then combine them with a *bag of words* approach. They pass them through a classifier to distinguish between benign and ransomware samples. They also use other statistical measures like entropy as features during classification.

Alam et al. [38] propose RAPPER, which uses a dynamic analysis approach to detect ransomware. Hardware performance counters (eg. branches taken, cache misses, etc.) are gathered into a time series. On normal operation of the system, an autoencoder is trained to reproduce the input with low reconstruction loss. After training, if a malware executes on the system, the auto-encoder will report a high reconstruction loss. As a part of feature engineering, they use fast Fourier transforms to convert the time series data into the frequency domain.

There are other approaches on these lines where frequency domain analysis has been used to detect ransomware with static and dynamic analyses. I have limited my discussion to just a few approaches.

My search has not revealed any papers where frequency domain analysis has been used to detect encrypted content in user data.

2.7 File Fragment Type Identification

File fragment analysis is the task of predicting the type of a file given a fragment of a file. For example, the type of the file can be jpeg, docx, encrypted, compressed, etc. Naturally, this task has overlaps with my proposal, and hence I review some selected literature around this task. With the exception of Penrose et al. [26] and Zhao et al. [37], these approaches do not make any attempt to distinguish between encrypted and non-encrypted content.

The most comprehensive review of file fragment analysis was done by Penrose et al. [26]. They note that in most cases, the available literature usually does not give any details about the nature, size, and source of the datasets, used, and hence, it is difficult to compare them. They further test the hypothesis that it is possible to distinguish

between encrypted content and compressed content by applying an efficient compression technique. A fragment of a compressed file should compress more than an encrypted file. This is because most compression algorithms do not achieve optimal compression as they aim to strike a balance between compression ratio and compression speed, and further compression of a compressed file may be possible. However, their approach is susceptible to adversarial attacks that employ entropy reduction after encryption (eg. base-64 encoding).

McDaniel and Heydari [39] introduce the concept of *byte frequency distribution* or BFD. This is a histogram of each byte value in a file fragment. They use the BFD as a signature of a file type. They also use a related measure *byte frequency cross-correlation* or BFC in a similar way. Since their original paper, there have been several papers that have used BFDs and BFCs in different ways to classify file fragments [40, 41] .

Veenman [42] uses BFD, entropy, and Lempel-Ziv complexity [43] (as a substitute for Kolmogorov's complexity [44]) along with a supervised learning algorithm to detect file types. Penrose et al. [26] incorrectly write that Kolmogorov's complexity is used by Veenman.

Sportiello and Zanero [45] use BFD, entropy, and a few other measures along with a support vector machine to classify file types.

Zhao et al. [37] use a battery of statistical tests on just seven very large files and argue that a combination of those tests gives the best discrimination between encrypted and non-encrypted content. However, the extremely small sample size makes their results unreliable.

However, Roussev and Garfinkel [46] argue that BFDs and BFCs are too simplistic and fail to take into consideration the inherent structure of different file types (like pdf, doc, tiff etc.). They suggest using methods that take advantage of the inherent structures of these files.

2.8 Research Gaps

From the literature review, a few gaps in current solutions emerge:

(G1) Most papers measure their performance against a very small set of files and file types and how well they perform in practice is unknown.

(G2) Most current approaches are easily bypassed by ransomware by employing evasive techniques like:

(a) Post-encryption entropy reduction (with techniques like base-64 encoding)

(b) Alternate encryption strategies like not encrypting the first few bytes or kilobytes, or encrypting every alternate 16 bytes.

(G3) These approaches completely fail to detect encryption in some file types like webp.

Chapter 3

Illustrating the Need For Better Features With an Attack

In this chapter, I illustrate the need for better features. I do this by demonstrating a trivial attack on a trained network that has learned to identify one kind of ransomware.

3.1 Methodology and Results

I took 50,000 files and created two copies of them.

1. I left the first copy unencrypted
2. I encrypted the second copy with AES-128 encryption

For each of the files above, I calculated the following features:

1. Shannon's entropy
2. Chi-square statistic
3. Monte Carlo simulation of π
4. Autocorrelation
5. File size

These features are frequently used in commercial anti-ransomware solutions.

I then trained a deep fully-connected neural network as described in Table 3.1.

Layer Name	Number of Neurons	Activation Function/Value
Dense-5	5	ReLU
dropout_2	5	0.2
Dense-4	4	ReLU
dropout_3	4	0.2
Dense-2	2	ReLU
Dense-1	1	Sigmoid

TABLE 3.1: Architecture of dense neural network

When I evaluated this with the dataset described earlier, I observed a test set accuracy of **0.947** and F1 score of **0.958**. The performance of this model was impressive.

I saved the model for future use.

3.2 The Attack Methodology

I then performed a trivial attack on this model. I took the encrypted files and encoded them in base-32 format. The contents of the file were still encrypted.

I calculated the same features for the encrypted and base-32 encoded files.

Once this was done, I tried to predict which files were encrypted using the saved model.

During this attempt, I observed that *both the accuracy and F1 scores were **0.0*** !!!

3.3 Analysis of the Attack

I further analyzed why this attack succeeded.

Figure 3.1 shows the values of Monte-Carlo simulation of π, Shannon's entropy, and autocorrelation for the three types of files:

1. Plain text files
2. Encrypted files
3. Files encoded in base-32 format after encryption

It is seen that the encrypted files form a cluster that is separated from the files that were base-32 encoded after encryption. The values of the plain text files are more distributed in space. With these three features, it will be difficult to learn a decision boundary that gives good accuracy of detection.

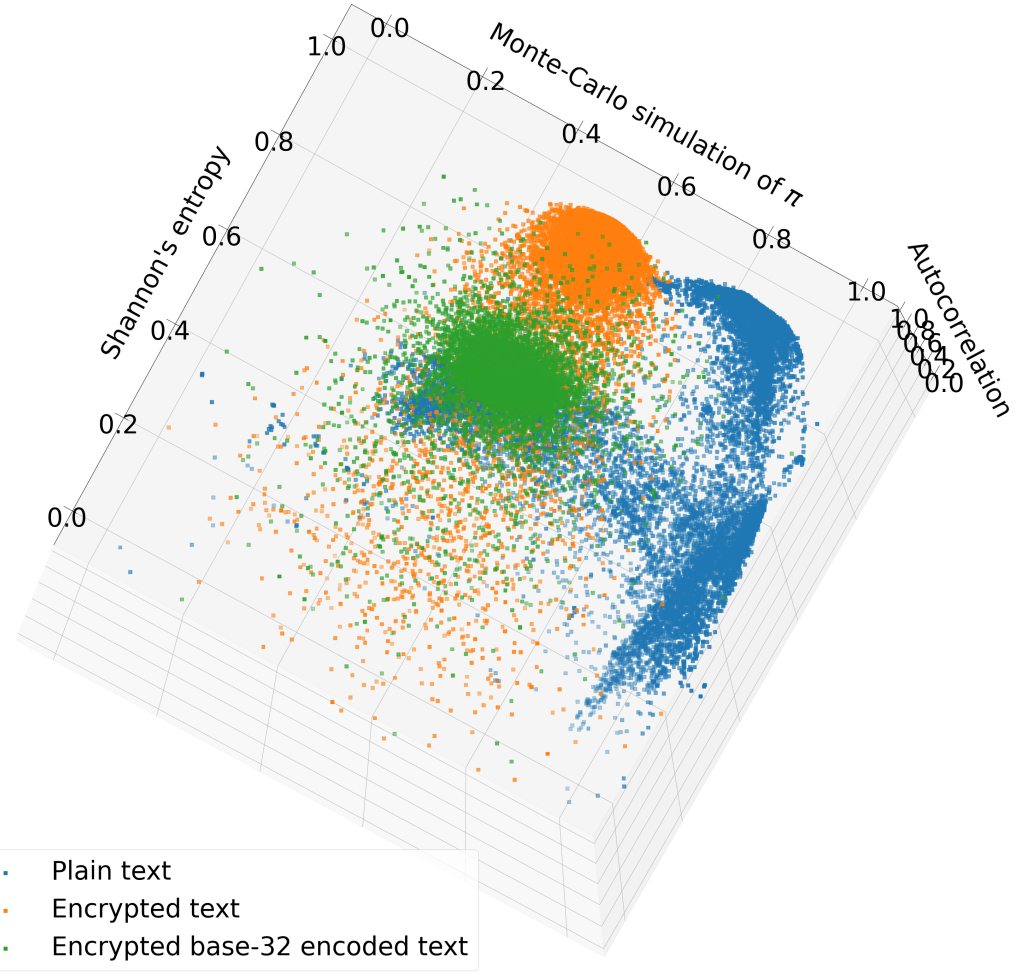

FIGURE 3.1: 3-D plot of three features for plain text, encrypted, and encrypted base-32 encoded files

Figure 3.2 illustrates the Shannon entropies of all the files. I observed that for plain text files, the entropies are widely distributed, but for encrypted files, all the entropies are very high and lie in a narrow band of values. The neural network learned a decision boundary using these values.

When the encrypted files were base-32 encoded, it resulted in a decrease in the entropy of the files to a smaller value. Thus, the decision boundary that was learned earlier classified these files as plain text. This is clearly understood from Figure 3.1.

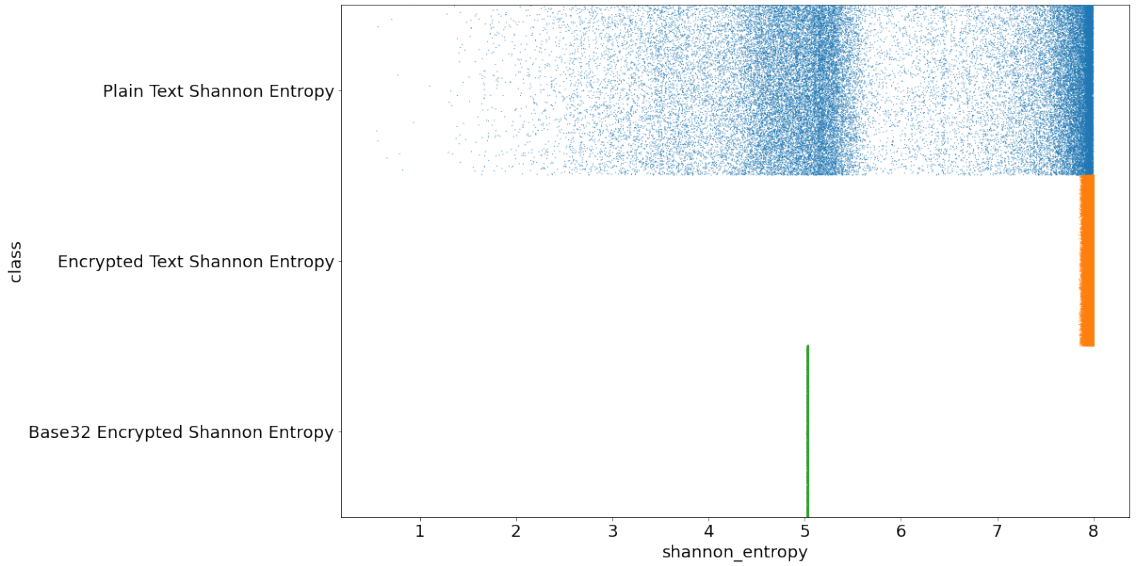

FIGURE 3.2: Shannon entropy for all files

Figure 3.3 illustrates the same effect for the Monte-Carlo simulation of π.

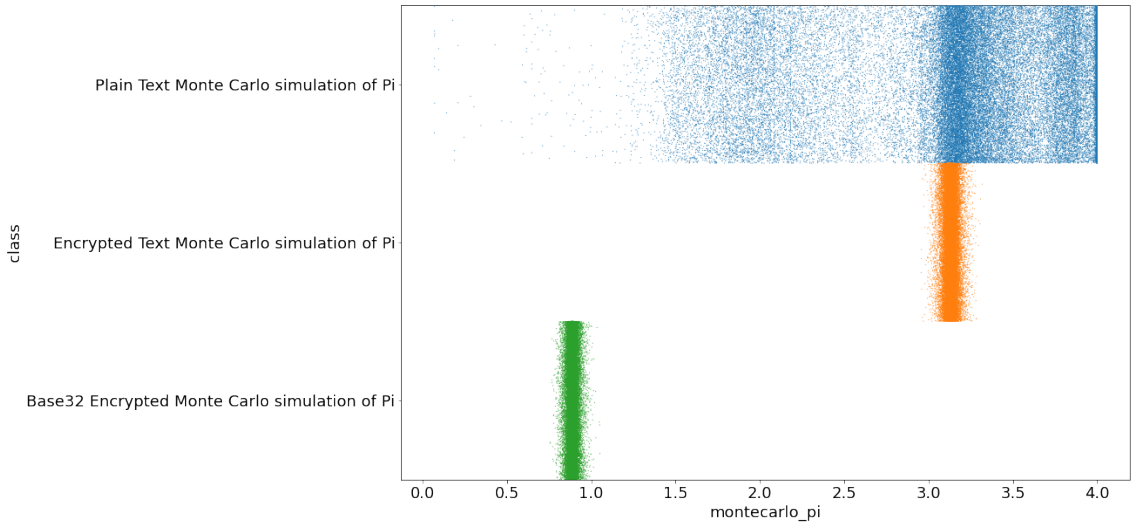

FIGURE 3.3: Monte-Carlo simulation of π for all files

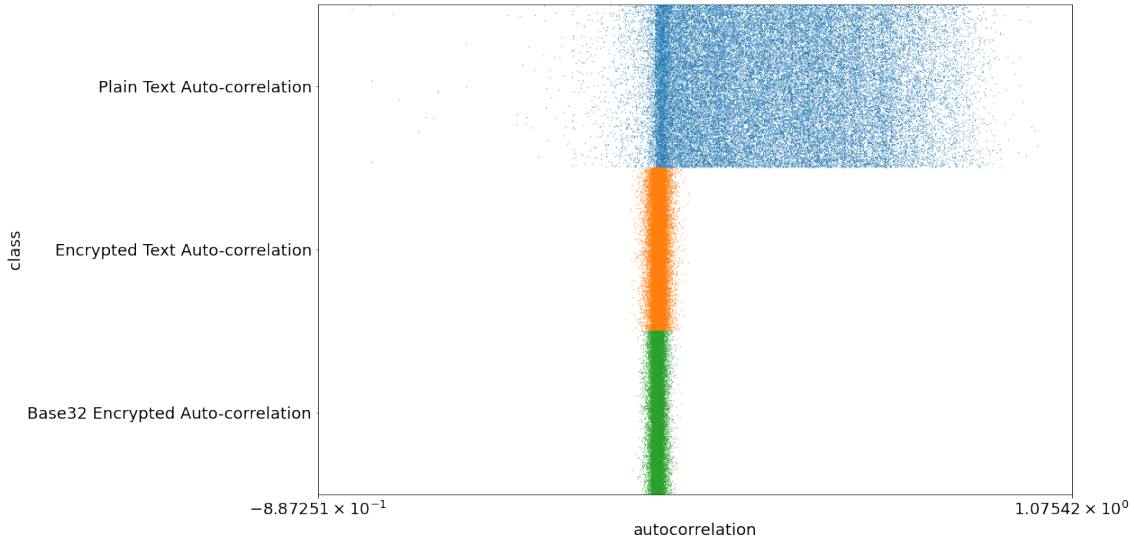

FIGURE 3.4: Autocorrelation for all files in *symlog* scale

Figure 3.4 shows the effect of base-32 encoding for the autocorrelation in symlog scale. I can see that the autocorrelation value is hardly affected by base-32 encoding. Why is that?

I can categorize the features into two broad types:

1. Features that capture information about the internal structure of a file (autocorrelation).
2. Features that do not capture any information about the internal structure of a file and rely solely on frequency counts.

The latter can easily be changed by changing the encoding scheme, but changing the former is more difficult. The greater the amount of information about the underlying structure of the file that the feature encodes, the more difficult it is to bypass detection systems that rely on such information by trivial means.

Therefore, I can see a need to include more features that capture a greater amount of information about the underlying structure of a file. Fourier transforms capture a great deal of information about repeated structures in a file, and intuitively, one can see why its inclusion can improve detection.

Chapter 4

Design

In Chapter 3, I demonstrated that there is a need for better features by showing an attack that bypasses detection by a trained model. In Section 2.8, I identified several gaps in the existing literature. To address (G1), I planned to train my models on a diverse set of file types and formats. The use of Fourier analysis, skewness, and kurtosis is promising to address (G2) and (G3). Furthermore, I also thought that autoencoders can be applied to the task of encrypted content detection to deal with file types I have not trained on.

In this chapter, I start with a general overview of the design, and follow it with the core objectives of the project. I include a discussion of entropy and complexity of compressed and encrypted data as this forms the theoretical basis of this work. I then provide a short description of how I plan to use Fourier transforms, skewness, and kurtosis in this work. I provide a brief description of how I planned to use autoencoders to further improve detection. I then describe the low level design of this work. Finally, I list the packages that were used while implementing the design.

4.1 Problem Definition

Since I feel that the most useful and most under-researched approach to ransomware detection is detecting encrypted contents in a file, I focus on this approach.

The problem I address here is to predict a file has potentially been encrypted by ransomware.

My main goal here is to find features that can be used with a multitude of machine learning models for accurate and robust prediction. The approach followed here is data centric rather than model centric.

4.2 Objectives

This project achieves the following objectives:

1. To define and evaluate a set of features for robust detection of encrypted files across a variety of file types, especially webp files.
2. To systematically study the effectiveness of the following measurements to discriminate between encrypted and plain content:
   (i) Fourier power spectrum density
   (ii) Kurtosis
   (iii) Skewness
   (iv) Central moments
3. To systematically study the effectiveness of the above methods to provide robust defense against anti-detection techniques used by ransomware like:
   (i) Not encrypting the first few bytes of a file
   (ii) Encrypting alternate blocks of n bytes
   (iii) Entropy reduction techniques
4. To differentiate between encrypted content, password protected content and compressed files using the features described earlier. This is necessary for reducing false positives.
5. To use, evaluate, and optimize a combination of the above features with traditionally used statistical measures along with an autoencoder network to provide a robust method to detect encrypted content
6. To improve speed of detection by using a sampling strategy without sacrificing too much accuracy

4.3 Requirements

As discussed earlier, most work on ransomware detection have not addressed the following:

1. Using large sample sets
2. Using a variety of file types
3. Testing against adversarial attacks by ransomware

The dataset sizes used in similar studies in the past are summarized in Table 2.1, and the mean dataset size is just 2252 samples. I used a base dataset of 90883 files, and used several variants of resulting in a total of over 500,000 samples. I also combined the NapierOne and GovDocs1 datasets to give a greater variety of file types than previous studies.

In this work I measured the effectiveness of several features, and they were tested against a variety of both simple and more sophisticated machine learning algorithms. The performance of these features was measured in all cases.

For all cases, the following cases were considered:

1. The plain text must contain a mixture of regular files and base-32 encoded files.
2. The encrypted text must contain regular encrypted files, as well as files encoded in base-32 format after encryption.
3. The dataset should contain files that are only partially encrypted.
4. The dataset should also contain password protected files (like password protected zips or office documents).
5. One test scenario should be to measure the effectiveness of trained models on unknown file types and datasets.
6. A large number of files should be used for evaluation to give a better estimate of real world performance. The mean sample size used in similar studies in the past is only about 2252 samples, and given the diversity of file types in the wild, I do not think that this is adequate. The size of the dataset I used has over 500,000 samples.

McIntosh et al. [34] have showed that entropy reduction techniques, like base-32 encoding, can be used to trivially bypass ransomware detection. Many plain text files are also base-32 encoded. Hence I decided to include base-32 encoded files in both the encrypted and plain text samples. Partial encryption is a technique many ransomware have used to bypass ransomware detection programs, and therefore, it is important to include them in the dataset. It is also important to test on datasets and file types that were not used during training. In the real world, a model will be expected to respond to file types it has not seen earlier and still perform robustly with a low false positive rate. The cases I chose are adequate to address the concerns that are addressed here.

4.4 Discussion of Choice of Features

The features discussed above can be broadly categorized into two types

1. Byte-frequency driven features or other statistical features that do not capture any information about the inherent structure of files. These include entropy measurements.
2. Features that capture the inherent structure of the files. Traditional measurements like autocorrelation capture some rudimentary information about the inherent structure of files, but much richer information about the inherent structure of files is captured by the Fourier transform.

Also, capturing many such feature provides resilience against adversarial attacks. This is because it may be possible for ransomware to tweak its encryption algorithm to change a few measurements and bypass detection, but it becomes more difficult for ransomware to bypass detection by changing multiple measurements which are arrived at using different methods.

In this work, I enriched the set of features that can be used to detect encrypted content in user data, and can be used for more robust detection across a variety of file types. This also makes adversarial attacks to bypass detection more difficult for ransomware, as engineering an encryption scheme to bypass a system that uses a diverse array of features is much more difficult than engineering an encryption scheme to bypass a system that uses limited features. As the number of features increases, the possible combinations of those features grow exponentially.

As far as I know, this is the first work to systematically study the application of Fourier transforms, skewness, kurtosis, and higher order central moments to the problem of detecting encrypted content in user data.

I also captured the same statistics for both the complete file and the extremities of the file. However, the measurements for the extremities were only used to prove that using selective measurements of certain fixed parts of the file makes it easier for ransomware to bypass detection by not encrypting those parts. For the rest of this work, only measurements for the complete file were used.

4.5 Theoretical Considerations

In this section, I visit some of the theoretical aspects that underpin this work.

4.5.1 General Information Theory

In information theory, *perfect secrecy* is a concept that specifies that after encryption of a plain text, the encrypted text gives no information at all about the original text. Shannon showed that for every perfectly secure cipher, the length of the key is at least as large as the length of the original message. As a result, perfect secrecy is impractical for use in the real world, and therefore, it must be relaxed. This has led to the definition of *computational security* which is based on a couple of relaxations of perfect secrecy:

1. The encryption is secure against computationally bounded processes that may try to break it.
2. The encryption may be broken with only a small probability.

Shannon [47] further postulates that two desirable properties of any encryption algorithm are:

1. *Diffusion*: A change in a single bit of the plain text should change many bits of the cipher text.
2. *Confusion*: Each bit of the cipher text should depend on many bits of the input plain text and key.

The above two properties directly imply that a good encryption algorithm should produce a cipher text that is seemingly random [48]. A random string can be understood as a string that has no apparent patterns.

Kolmogorov complexity [44] is useful to arrive at a more concrete definition of randomness of a string of bits. The Kolmogorov complexity of any string is the length of the smallest program that can produce the string as output. A string of bits can be considered random if the Kolmogorov complexity of the string is at least as long as the string itself.

However, it has been shown that the calculation of the Kolmogorov complexity is intractable, and hence other measures of randomness like Shannon's and Rényi's entropy and *Lempel-Ziv complexity* [43] are useful substitutes.

The same measures (Kolmogorov's complexity, Shannon's and Rényi's entropy and Lempel-Ziv complexity) also determine how much a string can be compressed, and as a result a scheme that relies on complexity alone may find it difficult to distinguish between encrypted and compressed text.

The Lempel-Ziv complexity measures the diversity of patterns in a string. If repeated patterns are found in a string, then it has a low Lempel-Ziv complexity, and as a result, the string can be compressed efficiently by replacing repeated occurrences of a pattern by tokens. The resulting compressed string will have a higher Lempel-Ziv complexity than the original uncompressed string. Lempel and Ziv [49, 50] extended the idea of Lempel-Ziv complexity to propose data compression schemes, which later became the basis of the zip algorithm.

Since a desirable property of an encrypted text is that it should not contain any patterns that can be used to derive any information about the original text, its Lemep-Ziv complexity must also be high. As a result, an encrypted string may not lend itself well to compression. Penrose et al. [26] test this hypothesis to distinguish between encrypted and compressed text. They observe that most compression algorithms strike a balance between compression speed and compression ratio, and that results in sub-optimal compression ratios which can be compressed further. They propose that since encrypted text cannot be compressed further, this can be used to differentiate between encrypted and compressed content. However, they fail to account for post-encryption entropy reduction techniques which can be used to work around a distinction based on compression ratio.

The presence or absence of patterns in any string of bits can also be detected by Fourier transforms. Furthermore, since Fourier transforms carry more information than a single measure like entropy, they can be used as a measure that is resilient against post-encryption entropy reduction techniques. Typically, texts with neat patterns show up as a few characteristic frequencies in the Fourier spectrum with other frequencies showing no activity. With encrypted text, a very noisy Fourier spectrum is observed with peaks in a multitude of frequencies. An example of this can be seen in Figure 4.3. Entropy reduction techniques do not significantly reduce the noise in the Fourier spectrum and this can be used as a defence against entropy reduction techniques.

Some audio and video file formats (like webp) which are compressed with lossy or lossless techniques naturally show high entropy, and entropy based metrics are not able to distinguish between them and encrypted content. Kurtosis and skewness are expected to be able to discriminate between the two. For any probability distribution, skewness measures the symmetry of the distribution and kurtosis measures the thickness of the tails. Encrypted content is expected to have thick tails and symmetric distribution, but the same is not expected to be true of plain text.

4.5.2 Skewness and Kurtosis

Skewness of a random variable is the third standardized moment and is described by equation 4.1.

$$\tilde{\mu}_3 = \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu_3}{\sigma^3} = \frac{\mathrm{E}\left[(X-\mu)^3\right]}{(\mathrm{E}\left[(X-\mu)^2\right])^{3/2}} = \frac{\kappa_3}{\kappa_2^{3/2}}. \tag{4.1}$$

μ_n is the n^{th} central moment, expressed by equation 4.2.

$$\mu_n = \mathrm{E}\left[(X - \mathrm{E}[X])^n\right] = \int_{-\infty}^{+\infty} (x-\mu)^n f(x)\,\mathrm{d}x. \tag{4.2}$$

Skewness measures the amount of asymmetry in a probability distribution function around its mean. Figure 4.1 illustrates two skewed distributions.

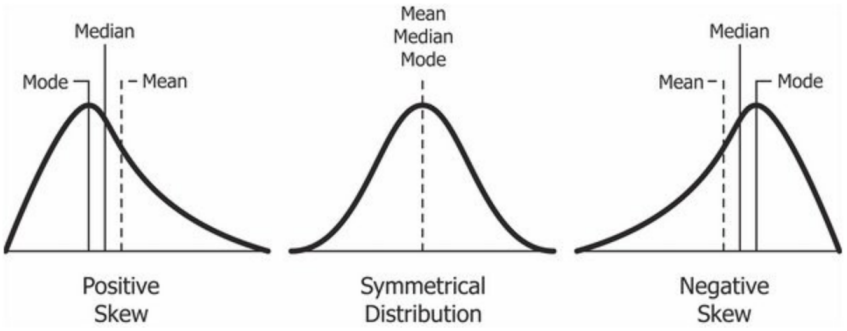

FIGURE 4.1: Example of skew in distributions. [Image source: Wikipedia]

The intuition here is that for any regular file, there is likely to be asymmetry around the mean of a probability distribution function. For example, if I take the English language, some characters are more used more often than other characters. However, encrypted files should show more symmetry around the mean.

Kurtosis is the fourth standardized moment, and is expressed in equation 4.3 where μ_4 is the fourth central moment, and σ is the standard deviation.

$$\mathrm{Kurt}[X] = \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\mathrm{E}\left[(X-\mu)^4\right]}{\left(\mathrm{E}\left[(X-\mu)^2\right]\right)^2} = \frac{\mu_4}{\sigma^4}. \tag{4.3}$$

Kurtosis measures how thick the tails are in a probability distribution function. Many papers that use kurtosis often remark that kurtosis measures the *peaked-ness* of a probability distribution function, but this is not necessarily true. Kurtosis really measures how thick the tails of a probability distribution function are [51, 52].

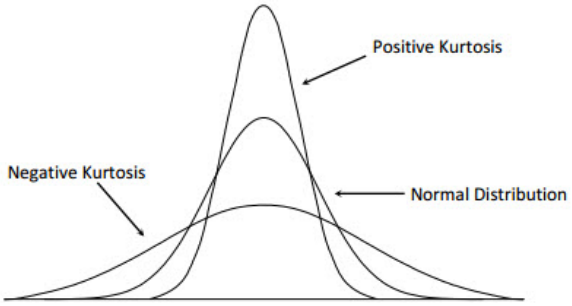

FIGURE 4.2: Illustration of different kurtosis values. [Image source: stats.stackexchange.com]

The intuition here is that regular files should show a fall-off at the tails, but encrypted content would show thicker tails as all characters are likely to be repeated an equal number of times. This effect should manifest itself in the kurtosis measure.

In addition to skewness and kurtosis, I also measured several central moments to see if these moments are beneficial in distinguishing between encrypted and plain content.

At this point, it is important to stress that the measures described above only deal with frequencies of bytes in a file and do not take into account any other structure within the file.

4.5.3 Other Statistics

The n^{th} central moment around mean is defined as follows:

$$\mu_n = \mathrm{E}\left[(X - \mathrm{E}[X])^n\right] = \int_{-\infty}^{+\infty} (x-\mu)^n f(x)\,\mathrm{d}x. \tag{4.4}$$

The first central moment is 0 and the second central moment is called the variance.

4.5.4 Entropy Measures

Entropy measurements are a set of scientific concepts that has several interpretations in science but have the same mathematical formula. In information theory, entropy measures the uncertainty in any event. According to Shannon's information theory, the information content of a highly probable event is less, and the information content of an event of low probability is high. Entropy has direct applications in several fields in computer science including communication and data compression.

The simplest measure of entropy that is used is Shannon's entropy. If an experiment has n possible outcomes $x_1, x_2, ..., x_n$ with probabilities $P(x_1), P(x_2), ..., P(x_n)$, then the Shannon entropy is given by equation 4.5.

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i). \tag{4.5}$$

If there are many events, and all the events have equal probability, then the entropy will be high. By contrast, if there are few events, and one or two of them have a high probability, then the entropy will be low. Therefore, encrypted content should have higher entropy than plain content.

Given a distribution of a stream of events that follows a probability density, entropy gives us a way to determine the smallest number of bits necessary to describe the stream of events without any loss of information.

Extropy is the dual of Shannon's entropy and is given by the equation 4.6. Unfortunately, extropy does not have any intuitive interpretation.

$$\text{Extropy}[X] = -\sum_{i=1}^{n} (1 - P(x_i)) \log(1 - P(x_i)). \tag{4.6}$$

Again, it is important to stress that these measures only take into account the frequencies or probabilities of different byte values in a file and contain no information about structures within a file.

Shannon's entropy is subadditive as specified in equation 4.7.

$$\mathrm{H}(A, B) \leq \mathrm{H}(A) + \mathrm{H}(B). \tag{4.7}$$

Furthermore, Shannon's entropy also satisfies strong subadditivity as expressed in equation 4.8.

$$\mathrm{H}(X, Y, X) + \mathrm{H}(Y) \leq \mathrm{H}(X, Y) + \mathrm{H}(Y, Z). \tag{4.8}$$

Alfréd Rényi proposed a more generalized form of entropy which is called Rényi's entropy.

$$\mathrm{H}_\alpha(X) = \frac{1}{1-\alpha} \log \left(\sum_{i=1}^{n} p_i^\alpha \right), \quad \alpha \in \mathbb{R}, \alpha \neq 1. \tag{4.9}$$

Like Shannon's entropy, Rényi's entropy is additive but does not satisfy strong additivity. Furthermore Rényi's entropy is parametrized by a real number α.

There is a special case of Rényi's entropy when $\alpha = 1$, where it yields a sum of Shannon Entropy and Kullback-Leibler divergence, as shown in equation 4.10.

$$\mathrm{H}(A, X) = \mathrm{H}(A) + \mathbb{E}_{a\sim A}\big[\mathrm{H}(X|A = a)\big]. \tag{4.10}$$

As $\alpha \to \infty$, Rényi's entropy is very heavily determined by the most probable event.

Constantino Tsallis has proposed another general entropy called Tsallis' Entropy

$$S_q(p_i) = \frac{k}{q-1}\left(1 - \sum_i p_i^q\right). \tag{4.11}$$

Here q is a real number and is called the *entropic index* and k is a constant.

Tsallis' entropy is non-additive but satisfies the following constraint.

$$S_q(A, B) = S_q(A) + S_q(B) + (1-q)S_q(A)S_q(B). \tag{4.12}$$

4.5.5 Fourier Analysis

As seen earlier, statistical measurements, like moments, and measures from information theory, like entropy, may be useful in detecting encrypted content by measuring the relative frequencies of different bytes, but they do not use any information regarding the structure of any file. Additionally, any inherent structure in files can be leveraged to determine whether they are encrypted or not. I used Fourier transforms for this.

Fourier expansion is a coordinate transformation in which any L-periodic function can be expressed as an infinite sum of orthogonal sines and cosines of increasing frequencies. One form of the Fourier expansion is given by equations 4.13, 4.14 and 4.15:

$$f(x) = \frac{A_0}{2}\sum_{k=1}^{\infty}\left(A_k \cos\frac{2\pi kx}{L} + B_k \sin\frac{2\pi kx}{L}\right). \tag{4.13}$$

where

$$A_k = \frac{2}{L}\int_0^L f(x)\cos\left(\frac{2\pi k}{L}x\right)\,dx, \quad k \in \mathbb{N}. \tag{4.14}$$

$$B_k = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{2\pi k}{L}x\right)\,dx, \quad k \in \mathbb{N}. \tag{4.15}$$

Fourier transform is a special case of Fourier series where $L \to \infty$. This is the case for general non-periodic functions. Equations 4.16 and 4.17 describe the Fourier transform and its inverse in the complex form.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\; e^{-2\pi i x\xi}\,dx, \quad \forall\; x \in \mathbb{R}. \tag{4.16}$$

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\; e^{2\pi i x\xi}\,d\xi, \quad \forall\; x \in \mathbb{R}. \tag{4.17}$$

The complex form of the Fourier transform is equivalent to a sum of sines and cosines; this follows from Euler's formula which is described in equation 4.18

$$e^{2\pi i\xi x} = \cos(2\pi\xi x) + i\sin(2\pi\xi x). \tag{4.18}$$

Conceptually, one can think of Fourier transforms and their inverses as taking a function in one set of coordinates and expressing it as a sum of functions in another set of coordinates. In signal processing, functions in the time domain are transformed into a function in the frequency domain. Equation 4.19 shows the relationship between a Fourier transform and its inverse.

$$f(x) \overset{\mathcal{F}}{\longleftrightarrow} \hat{f}(\xi). \tag{4.19}$$

In my design, I use the fast Fourier transform. The term is really a misnomer and what is actually calculated is the Fourier Series for an L-periodic function, rather than the case where $L \to \infty$. The fast Fourier transform is expressed in equation 4.20.

$$X_k = \sum_{n=0}^{N-1} \hat{x}_n e^{-i2\pi kn/N} \qquad k = 0, \ldots, N-1;\ \hat{x}_k \in \mathbb{C}. \tag{4.20}$$

As it can be seen, the fast Fourier transform returns a vector of complex numbers. Since every complex number has two parts, it is sometimes useful to take the product of every term of the FFT with its complex conjugate to arrive at a single real number. This is called the *power spectral density* and the same is expressed in equations 4.21 and 4.22.

$$S_{xx}(f) = \lim_{T\to\infty} \frac{1}{T}|\hat{x}_T(f)|^2. \tag{4.21}$$

$$|\hat{x}_T(f)|^2 = \mathcal{F}\left\{x_T^*(-t) * x_T(t)\right\}. \tag{4.22}$$

The power spectral density can often be very noisy, and Welch's method can be used to smooth the noise. This method involves the following:

1. The vector describing the segment of the function is divided into several overlapping windows. The number of such windows and the amount of overlap are parameters.
2. A *taper* is applied to each window so that the extremities of the window is 0. This is achieved by multiplying with another function. There are several such *window functions*.
3. The power spectral density for each window is calculated separately.
4. All the power spectral densities for all the windows are averaged for the final power spectral density.

If the file is treated as a function $f(i) = v$ representing a byte value v at index i, Fourier analysis can be used to detected any repeated pattern in a sequence of bytes. When passed through a Fourier transform, most files show peaks in certain frequencies. By contrast, a good encryption scheme should appear random and should not have easily identifiable patterns. Hence they should not show peaks at certain frequencies and their spectrum must be spread out wide. This can be used to distinguish between encrypted and plain content. An example of the power spectrum of an encrypted and regular file is presented in Figure 4.3.

In my search through the existing literature, I could not find any instance where Fourier transforms have been used to differentiate between encrypted and plain text files.

4.5.6 Autoencoders

I used an autoencoder network to distinguish between encrypted and plain content using the reconstruction loss. The idea here is to train an autoencoder on regular files using the features mentioned earlier, and train it to reconstruct those features accurately. Once the autoencoder learns to reproduce them, it should be able to reconstruct these features with low reconstruction error. Thereafter, if the autoencoder is presented with an encrypted file, it should result in a large reconstruction loss. A suitable cut-off can be chosen and any reconstruction above the cut-off can be classified as encrypted.

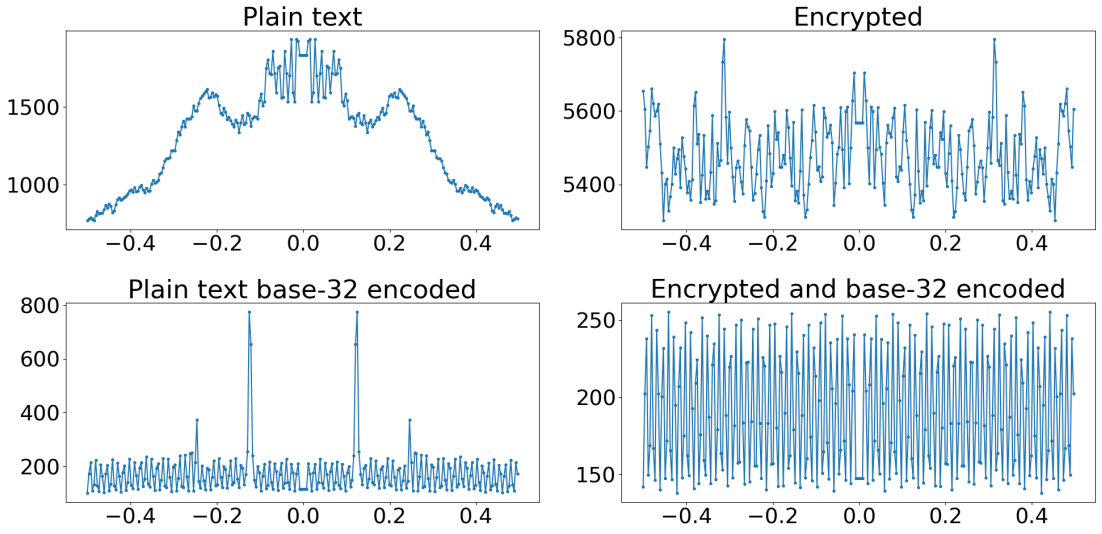

FIGURE 4.3: Power spectral density with Welch's method for Mary Wollstonecraft Shelley's *Frankenstein; Or, The Modern Prometheus*

4.6 Low Level Design

This section describes the design of this work. As I have noted in Chapter 2, the traditional approach is to gather statistics of files, and use them with a secondary machine learning classifier to predict whether they are encrypted or not. I have already discussed the theoretical aspects of the measurements I use in Section 4.5. I now explain in more detail how I used the measurements I discussed earlier.

1. Copies of the individual files from the dataset are created and transformed in the following ways:
   (a) Plain text
   (b) AES-128 encryption of the full file
   (c) 3DES encryption of the full file
   (d) AES-128 encryption of the file with the exception of the first 128 bytes
   (e) AES-128 encryption of the file with the exception of the first 56 bytes
   (f) AES-128 encryption of every alternate 16 bytes of the file
2. For both the plain text and encrypted text, two versions are used as follows:
   (a) The contents of the file as they are
   (b) Base-32 encoded contents of the file
3. For each of these files, several metrics were gathered to form a training and test dataset. The following metrics were used.

(a) Shannon's entropy
(b) Autocorrelation
(c) Chi-square
(d) Monte-Carlo simulation of π
(e) Skew
(f) Kurtosis
(g) Central moments ($n = 2$ through $n = 15$)
(h) Tsallis' entropy ($q = 1$ through $q = 10$)
(i) Rényi's entropy ($\alpha = 1$ through $\alpha = 10$ and $\alpha = \inf$)
(j) Power spectrum density with the Welch's method [53]. Two separate measurements were used:
    i. Treating each byte as a data point
    ii. Combining every four consecutive bytes into a single integer to represent a data point

    For each of the above, the following was calculated:
    i. Autocorrelation
    ii. Mean
    iii. Standard deviation
    iv. Chi-squared metric
    v. Central moments ($n = 2$ through $n = 15$)

4. Create three sets of features:
    (a) **Baseline**, which includes Shannon's entropy, chi-square, autocorrelationand Monte-Carlo simulation of π.
    (b) **Advanced**, which includes central moments, skew, kurtosis, Rényi's and Tsallis' entropy.
    (c) **Fourier**, which includes the raw Fourier power spectral density and the statistics of the spectral density. The autocorrelation, mean, standard deviation, and central moments for the raw Fourier power spectral density values are also calculated.
5. All combinations of the three feature sets described in 4 were evaluated against a host of methods.
6. Several models were trained with all the above features to determine which combination of feature sets performs the best.
7. An autoencoder was trained with reconstruction loss, and trained on plain text files and each combination of the feature sets. The Area under the curve was measured to see if it is possible to choose a cut-off that can lead to effective classification.

8. Similar to 7, an autoencoder was used to see if it can be used for classification. In this case, the autoencoder was trained on encrypted files.

The following measurements were used to evaluate the results.

1. The accuracy of a binary classification between the two classes: encrypted and plain files were measured. The primary metrics used for evaluation were the following:

$$\begin{aligned} Precision &= \frac{TP}{TP+FP}. \\ Recall &= \frac{TP}{TP+FN}. \\ F1 &= 2 \cdot \frac{Precision \times Recall}{Precision + Recall} = 2 \cdot \frac{TP}{TP + \frac{1}{2} \cdot (FP + FN)}. \end{aligned} \tag{4.23}$$

   Apart from this, accuracy was also measured.

2. The same measurements were performed for the following encryption strategies:
   (a) Encrypting entire files
   (b) Encrypting files but not the first 56 or 128 bytes
   (c) Encrypting alternate blocks of 16 bytes
   (d) Encoding in base-32 format after encryption

As discussed earlier, the goal of this work is not to produce a single model to predict ransomware; rather the goal of this is to suggest features that can enable all models to learn to detect ransomware more effectively. In many cases, system performance and speed considerations along with restrictions on interpretability may dictate that only simple models may be used. In other cases, more sophisticated models may be used for greater accuracy. Hence, it becomes important to test against both simple and more sophisticated methods. I chose the following models to test against the following models.

1. Logistic regression
2. Random forest classification
3. Densely connected artificial neural networks
4. Autoencoders

I feel that logistic regression is the best test to compare between different sets of features, as it is one of the simplest machine learning models, and can best highlight how good a decision boundary a particular feature can give. Also, logistic regression is least

susceptible to over-fitting and more sophisticated models can be prone to over-fitting, thereby making observations unreliable especially when the dataset is not representative of files in the field.

I used the following encryption schemes as described in Table 4.1.

Name	Description
v1	Encrypt the entire file
v2	Do not encrypt the first and last 128 bytes Encrypt every alternate 16 bytes of the remaining bytes
v3	Do not encrypt the first and last 56 bytes Encrypt every alternate 16 bytes of the remaining bytes

TABLE 4.1: Encryption Schemes

4.7 Packages Used

I used the following packages:

1. **Scikit-learn** version 1.0.1 was used for its implementations of random forest and logistic regression. Scikit-learn is a free software and is the most popular library for traditional machine learning algorithms for classification, regression and clustering. It is mature library written in python, cython, C and C++, and is under active development.
2. **TensorFlow** version 2.7.0 was used to implement artificial neural networks. Tensorflow is one of the most popular libraries for training and inference with deep neural networks. TensorFlow was developed by the Google Brain team, and was later open sourced. It is under active development.
3. **SciPy** version 1.7.2 was used for some computations. SciPy is one of the most popular python libraries for scientific computation. It has modules for optimization, linear algebra, statistics, fast Fourier transforms, signal processing and image processing. SciPy is mostly written in Python, Fortran, C and C++.
   (a) The **scipy.stats** package was used to calculate some statistics.
   (b) The **scipy.signal** package was used to calculate the Fourier power spectral density.
4. The **discrete information theory (dit)** python package (version 1.2.3) was used to calculate several general entropies. The dit has implementations for calculating various generalized entropies and rate-distortion curves.

5. The **PyCryptodome** package (version 3.12.0) was used to encrypt data. PyCryptodome is a popular library for low-level cryptography. PyCryptodome implements a large number of encryption algorithms, cryptographic hashes and public key cryptography standards.

In this chapter, I have defined the problem and listed the requirements and my objectives. I have also discussed my choice of features and the underlying theory. The variations of the dataset, the low level features, and the evaluation criteria was explained in the low level design. Chapter 5 describes the implementation and evaluation of this design.

Chapter 5

Implementation and Evaluation of Results

The implementation of this work was done in several agile sprints, with each sprint building up on the previous iteration. We present the results in the same fashion. We first start with an overview of what we found in each iteration. We follow this by a description of the dataset, and finally, we describe the methodology of each sprint, and extensively discuss the results from each iteration.

Since our goal was to measure the effectiveness of different features and their combinations, we ran experiments with logistic regression, which a very simple and unsophisticated classifier. We did this so that the effectiveness of the features, rather than the model can be stressed.

In addition to this, we also used a random forest classifier for two reasons:

1. To calculate relative importance of features, since random forest gives us the ability to capture the importance of each feature.
2. To give an idea of how effective a more sophisticated classifier will be with newly introduced features.

5.1 Overview of Sprints

In the first sprint, we achieved a simple baseline with the most common statistics that are used in commercial software: entropy, chi-square, autocorrelation and Monte Carlo simulation of π, kurtosis, and skew. Our key findings here are that, taking the statistics

of the extremities of the files in addition to the entire file can boost accuracy and F-scores by a very large amount. However, this also implies that any ransomware aware of such selective measurements of statistics of extremities of files can avoid detection by simply not encrypting the first and last few bytes of any file. This evasive technique is quite common among ransomware in the field [10, 12]. Thus, for the rest of the iterations, we did not use statistics of the extremities of the files separately, and relied only on statistics for the full file. Furthermore, we do not rely on any statistics that can very simply identify a file (like file size, or extension), since any such features can be trivially bypassed by ransomware by either changing the file extension or by padding zero bytes at the very end.

In the second sprint, I investigated and evaluated the effect of advanced statistics and Fourier power spectral density with AES-128 encryption. The third sprint evaluated the same with 3DES encryption. In Sprint 4, I evaluated the performance of the the features with unseen file types, archives, and password protected files. The goal of Sprint 5 was to evaluate the use of autoencoders. In Sprint 6, I evaluated the effectiveness of different feature sets with a dense neural network. The goal of Sprint 7 was to evaluate the performance with files that were encrypted with real ransomware. Sprint 8 dealt with using a sample of bytes from a file for detection, instead of using the full contents of the file.

5.2 Dataset Description

As noted in Chapter 2, comparison of existing approaches to ransomware detection is difficult because most of the literature does not mention the source or nature of their datasets in any detail.

Only a few papers use standard datasets, and among them, GovDocs1 [54] is the most popular. The full GovDocs1 dataset is quite large, and most papers only use a subset of the dataset. Pont et al. [33] pointed out that the diversity of the GovDocs1 dataset is quite limited and is not representative of real world files.

NapierOne [55] is a new much larger dataset that is based on GovDocs1, but has a greater diversity of file types. However, this dataset is yet to gain popularity. At the time of writing this report, the paper had no citations.

The OPF Corpus [56] is another dataset that has been used by a few studies, but is not very popular.

I found one use of audio compression files from Nigel Coldwell's website [57].

In this project, I primarily used the GovDocs1 dataset as it is the one used by most studies. Furthermore, I used a larger and more representative set of files from the GovDocs1 and NapierOne datasets than all studies I have reviewed. Similar studies have used a mean dataset size of 2252 samples, and a median dataset size of 683 samples. The sizes of datasets used in similar studies are summarized in Table 2.1.

As Pont et al. [33] pointed out, it is important to increase the diversity of the dataset by incorporating files that have high entropy, like webp files. I incorporate their suggestions by converting all jpeg and png files to webp files of various qualities.

5.2.1 GovDocs1

The GovDocs dataset is distributed in several ways, one of which is a set of 1000 zip files, each containing roughly 1000 files [58]. I randomly selected 50 such zip files, which gave a total of 49,228 files. Figure 5.1 illustrates the number of files of different types.

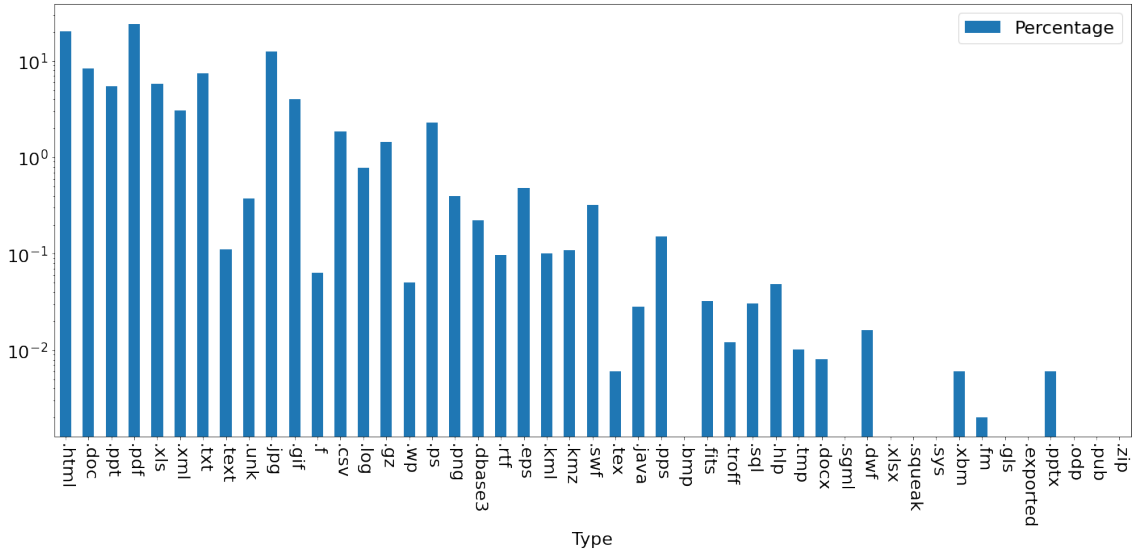

FIGURE 5.1: Number of each type in the downloaded dataset.

I augmented this set of files by converting each image to five webp files of the following qualities: 0%, 25%, 50%, 75%, and 100%. Figure 5.2 shows the number of files of each type in the augmented dataset, and Figure 5.3 shows the same in the log scale for a better view. Figure 5.4 presents the relationship between file size and number of files.

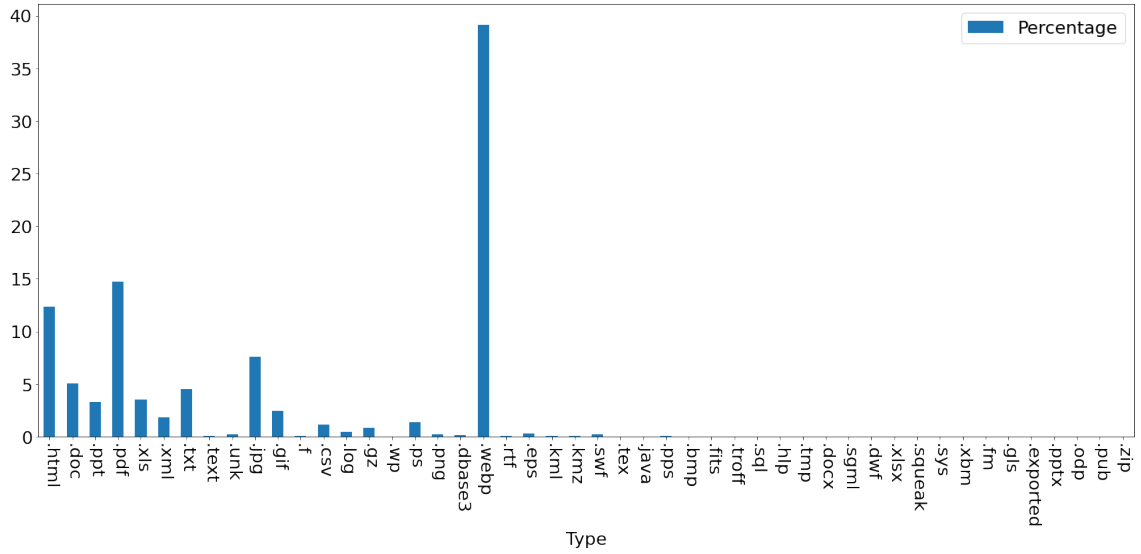

FIGURE 5.2: Number of each type of file in the augmented dataset.

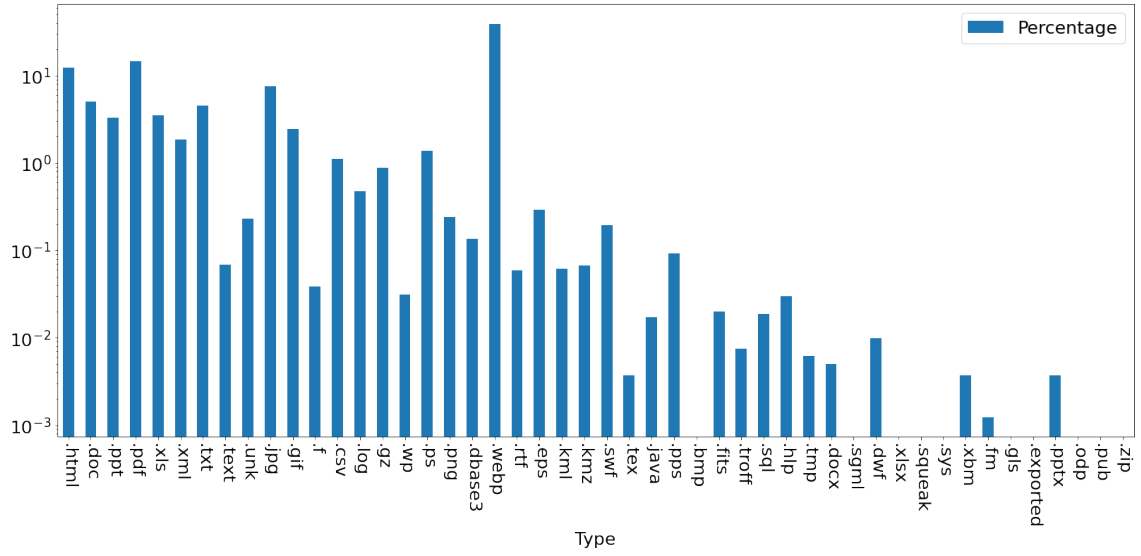

FIGURE 5.3: Number of each type of file in the augmented dataset in the log scale.

After augmentation, I finally had 80,853 files. I took this set of files, and transformed them in the following ways to create several different sets of files:

1. Unencrypted files
   (a) Any file that is smaller than 1500 bytes was appended unto itself repeatedly till it became at least 1500 bytes long. I call it the expanded plain-text.
   (b) The files were converted to base-32 encoding. I call this base-32 plain-text.
2. Encrypted files
   (a) I encrypted the expanded plain-text files in their entirety using AES-128 encryption. I call this *v1 encryption*.

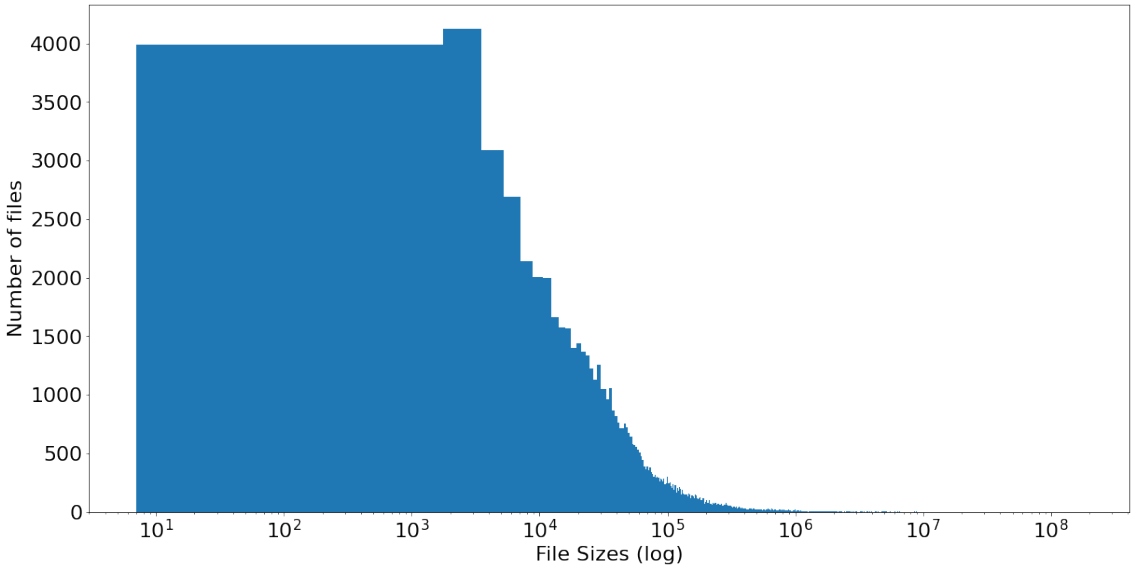

FIGURE 5.4: File sizes in the log scale vs number of files.

(b) I encrypted the expanded plain-text files in their entirety using AES-128 encryption, and then encoded it in base-32 format. I call this *v1-base32*.

(c) I encrypted the expanded plain-text files using the following scheme:

   i. The first and last 128 bytes of the file were not encrypted.
   ii. Of the remaining bytes, every alternate 16 bytes was encrypted.

   I call this *v2 encryption*.

(d) v2 encrypted files were converted to base-32. I call this *v2-base32*.

(e) I encrypted the plain-text a with similar encryption scheme as v2 encryption, except that I left the first and last 56 bytes unencrypted. I call it the *v3 encryption* scheme.

(f) A corresponding base-32 encoded version of v3 encrypted files was also created.

After encryption, I ended up with a combined dataset that has 485,118 samples.

The process was then repeated for 3DES encryption.

My rationale for expanding files by repeating them unto themselves is that for very small files statistical measures will not be meaningful. I had two choices here, either to discard files smaller than a threshold. If I did that, to make up for the dataset, I would have had to download more files. Instead, I chose to repeat smaller files unto themselves to create larger files.

There are several key things to note here:

1. Repeating the files unto themselves does not affect any plain-text statistics, since the relative frequencies of the bytes are unchanged.

2. Repeating files unto themselves does not affect the Fourier transform, because the Fourier transforms of any finite segment of a curve works by repeating the same segment to infinity on all sides.

3. Repeating a file before and after encryption produces very different results. Repeating after encryption just repeats the same encrypted text again and again, but repeating before encryption ideally produces no repeated text in the encrypted file. This is because of the property of diffusion of encryption algorithms that is explained in Chapter 4.

The primary objective of base-32 encoding was to test how resilient our methods are to entropy reduction techniques employed by ransomware. The purpose of the v2 encryption scheme was to test against adversarial attacks, and a similar attack was used by the LockFile ransomware [10]. I used the v3 encryption scheme to demonstrate the effects of a different choice of the unencrypted extremities of a file. In any run, either the v2 was used or the v3 was used, but not both. In total, I ended up with 6 sets of plain-text files, and 4 sets of encrypted files (counting either v2 or v3, but not both).

5.2.2 NapierOne

The GovDocs1 dataset has been the most popular dataset, but it has limited file types. More importantly, it has not added new file types and therefore, it is not representative of files in the wild. NapierOne is a new dataset that is maintained by the School of Computing at the Edinborough Napier University [59]. This dataset has a more diverse set of file types. I used the NapierOne tiny set, which contains a number of zip files, each of which has 1000 samples. Table 5.1 lists the file types and counts for the NapierOne tiny dataset. Figure 5.5 shows the sizes of the files in the NapierOne dataset.

File Type	Count	File Type	Count
7ZIP-BZIP2	1000	7ZIP-ENCRYPTED	1000
7ZIP-HIGH-COMPRESSION	1000	7ZIP-LZMA	1000
7ZIP-LZMA2	1000	7ZIP-PPMD	1000
APK	1000	BIN	1000
BMP	2000	CSS	1000
CSV	1000	DLL	1000
DOC	2000	DOC-PASSWORD	1000
DOCX	2000	DOCX-PASSWORD	1000
DWG	2000	ELF	1000
EPS	2000	EPUB	1000
EXE	1000	GIF	2000
GZIP	1000	HTML	1000
ICS	1000	JAVASCRIPT	1000
JPG	6000	JSON	1000
MKV	1000	MP3	1000
MP4	2000	ODS	1000
OXPS	1000	PDF	2000
PDF-PASSWORD	1000	PNG	6000
POWERSHELL	1000	PPT	2000
PPT	1000	PPTX	2000
PPTX	2000	RANDOM-PURE	1000
RANSOMWARE-DHARMA	1000	RANSOMWARE-MAZE	1000
RANSOMWARE-NETWALKER	1000	RANSOMWARE-NOTPETYA	1000
RANSOMWARE-PHOBOS	1000	RANSOMWARE-RYUK	1000
RANSOMWARE-SODINOKIBI	1000	RAR	1000
SVG	2000	TAR	1000
TIF	2000	TXT	1000
WEBP	9000	XLS	2000
XLS-PASSWORD	1000	XLSX	2000
XLSX-PASSWORD	1000	XML	1000
ZIP-BZIP2	1000	ZIP-DEFLATE	1000
ZIP-ENCRYPTED	1000	ZIP-HIGH-COMPRESSION	1000
ZIP-LZMA	1000	ZIP-PPMD	1000
ZLIB	1000	**TOTAL**	**10000**

TABLE 5.1: File types and counts in NapierOne

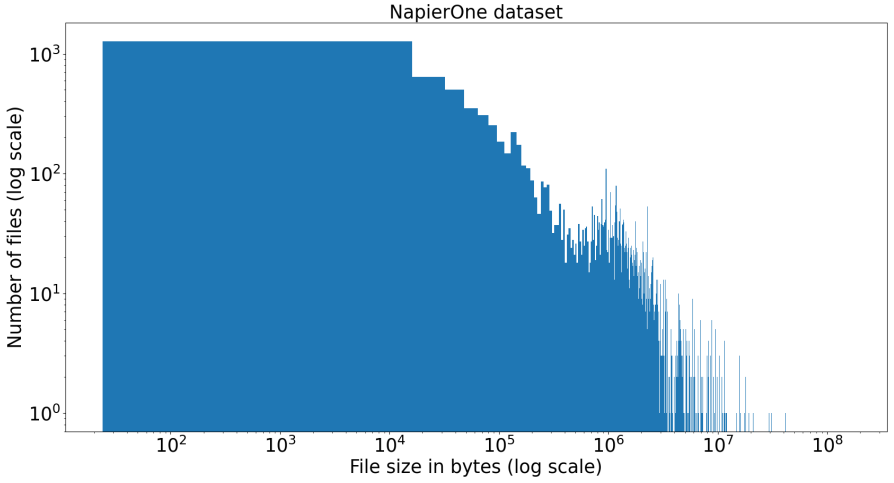

FIGURE 5.5: Sizes of files in the NapierOne dataset

5.3 Sprint 1: Establishing a Baseline

The goal of this sprint was to setup a baseline with the features that are most commonly used in commercial ransomware detection solutions.

5.3.1 Methodology and Implementation

The goal of this sprint was to create a baseline with the most common statistics used in commercial ransomware detection solutions, namely:

1. Autocorrelation
2. Shannon entropy
3. Monte Carlo simulation of π
4. Chi-square statistic

In this sprint, I compared the scenarios where:

1. Only the statistics of the full file are used
2. Statistics for parts of the file are used as follows:
   (a) First 128 bytes of the file (head extremity)
   (b) Last 128 bytes of the file (tail extremity)
   (c) The entire file

Furthermore, it has been suggested that kurtosis and skew might help in achieving higher accuracy of detection, so I repeated the same experiments above with kurtosis and skew [33].

Since I was only establishing a baseline at this point, I did not perform a cross-fold validation, and only performed a simple validation. Later sprints provided cross-fold validation for robust numbers. However, these numbers are still useful to give us pointers for the rest of the project.

To create the training and test dataset for this sprint, statistics were gathered for the following copies of the files:

1. The original set of files from the GovDocs1 dataset, unchanged
2. The set of files from the GovDocs1 dataset encrypted with AES scheme

Statistics were calculated for:

1. The entire file
2. The first 128 bytes of the file (head extremity)
3. The last 128 bytes of the file (tail extremity)

5.3.2 Results and Evaluation

The overall results of this run are summarized in Table 5.2.

The **best** and **worst** performances are highlighted. This scheme is followed throughout this report.

Algorithm	Kurtosis & Skewness	Extremity Measurement	Accuracy	Precision	Recall	F1
Logistic regression	NO	NO	**0.669**	**0.613**	**0.899**	**0.729**
Logistic regression	YES	NO	**0.669**	**0.613**	**0.899**	**0.729**
Logistic regression	NO	YES	0.765	0.728	0.837	0.779
Logistic regression	YES	YES	**0.769**	**0.737**	**0.831**	**0.781**
Random forest	NO	NO	0.650	0.634	**0.695**	**0.663**
Random forest	YES	NO	**0.649**	**0.630**	0.706	0.665
Random forest	NO	YES	0.748	0.749	0.738	0.744
Random forest	YES	YES	**0.750**	**0.752**	**0.741**	**0.746**

TABLE 5.2: Summary of results from Sprint 1

Broadly, the following were observed:

1. Measuring extremities improves F1 score and accuracy of detection by about 0.1. However, the corollary to this is that malware can easily bypass any detection that relies on extremity measurements by not encrypting extremities of a file. This is discussed in more detail below.

2. It is noticed that in some cases, random forest performs worse than logistic regression. This may be because random forest could be over-fitting the training data.

With the runs with random forest, I captured the feature importance measures, and they are shown in the graphs below in Figures 5.6, 5.7, 5.8, and 5.9.

From Table 5.2, it is clear that measuring the statistics of the extremities of a file, in addition to the statistics for the entire file, can give a giant boost in accuracy and F1 score. Here, by extremity, I mean the first and last few bytes of a file.

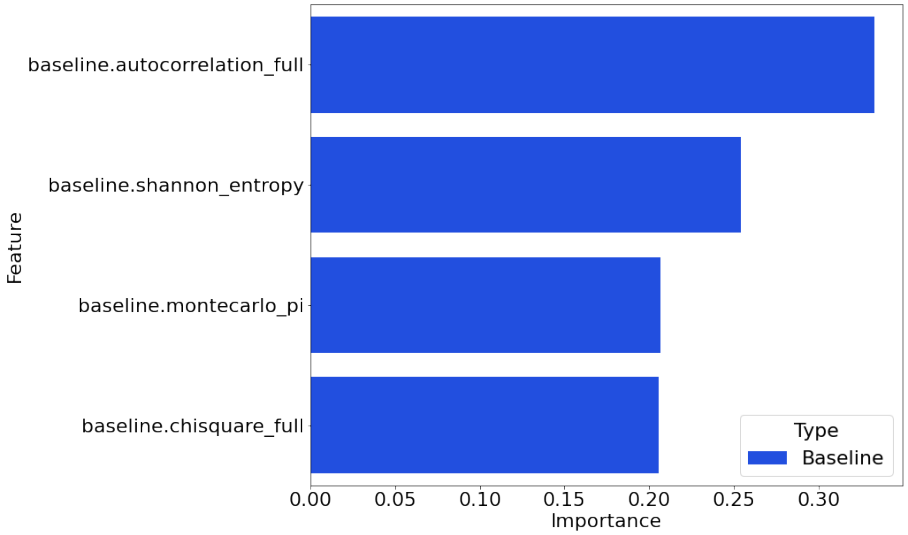

FIGURE 5.6: Random forest feature importances of baseline statistics when extremities are not measured separately

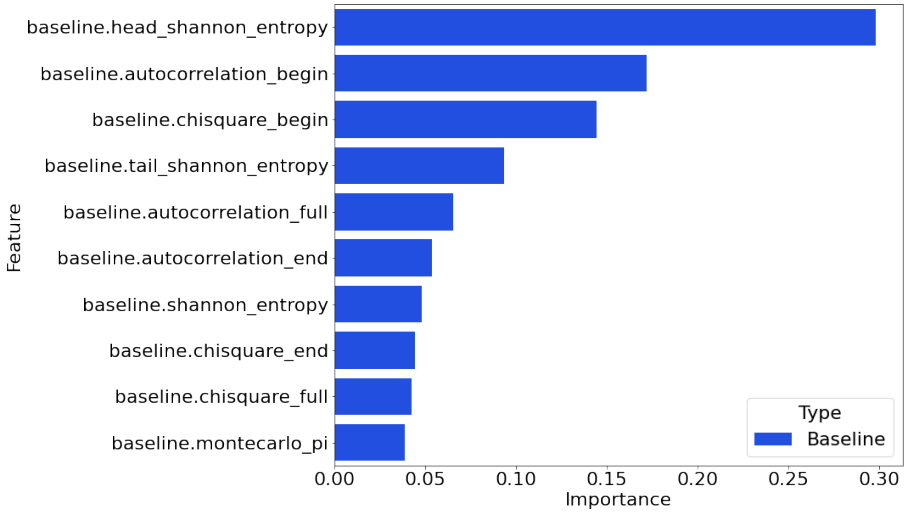

FIGURE 5.7: Random forest feature importances with baseline statistics when extremity statistics are measured separately

The first few bytes of many file formats have a very predictable header, and that limits the statistics to a narrow band of values. This makes it easier to use that to determine that a file is not encrypted.

However, from the above, it also follows that any malware that is aware of this technique can easily avoid detection trivially by not encrypting the first and last few bytes of the file. Any method that relies on the extremity statistics for detection is likely to be easily bypassed [35, 36]. From the random forest classifier, I captured the feature importances. The degree of reliance on extremity statistics is illustrated in Figures 5.7 and 5.9.

The other observation that can be made from Table 5.2 is that when kurtosis and skewness are combined only with baseline statistics, they do not have a significant impact

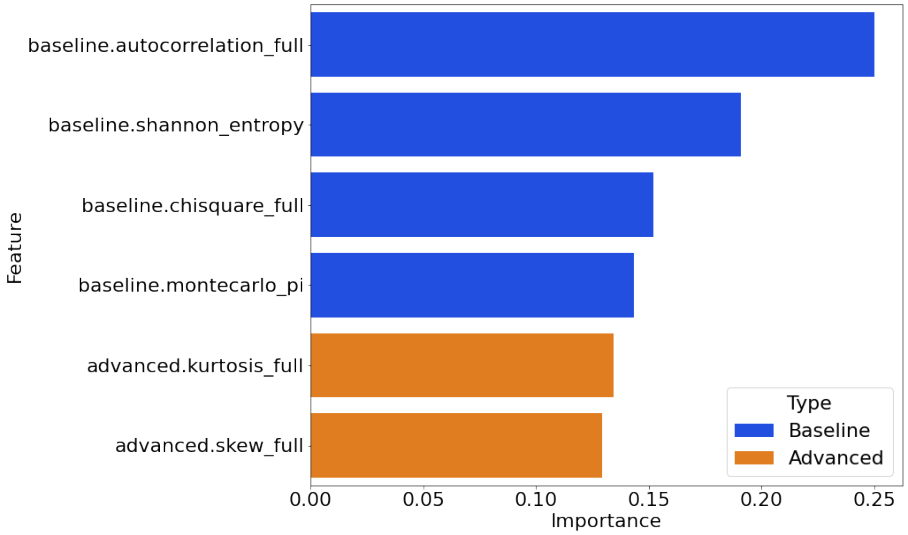

FIGURE 5.8: Random forest feature importances with baseline statistics, kurtosis, and skew

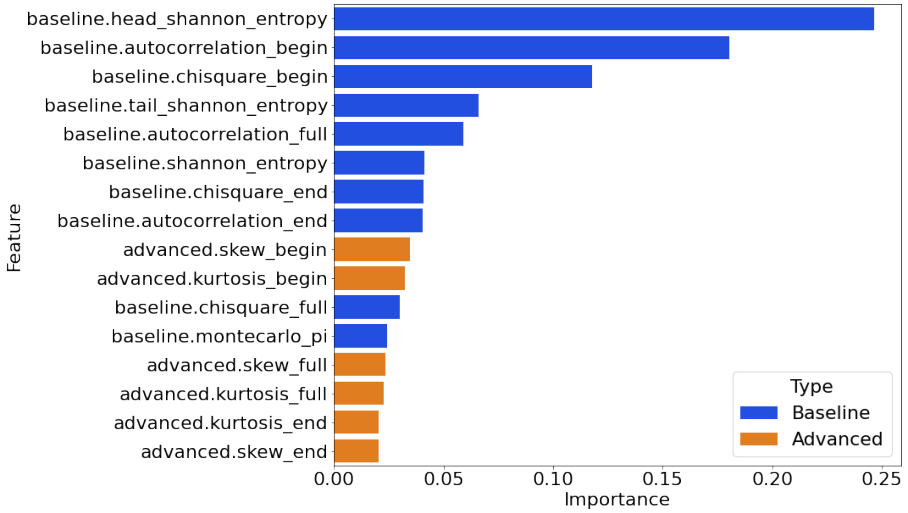

FIGURE 5.9: Random forest feature importances with baseline statistics, kurtosis, and skew when extremity statistics are measured separately

on accuracy. However, later in Sprint 2, I observed that kurtosis and skew perform quite well when combined with other advanced statistics.

In future sprints, I avoided using any features that can be trivially bypassed by ransomware. I avoided using extremity statistics. I also avoided using the filenames or extensions of a file, as they can be easily bypassed by renaming the files. The file size was also avoided as a feature because it can be easily modified by padding a file with extra bytes.

5.3.3 Review of Key Takeaways

The key findings of the sprint are summarized below.

1. Random forest is better able to detect encryption more accurately than logistic regression.
2. Measuring extremities of the files separately results in around an increase of about 0.1 in detection accuracy and F1 score. This could be because the headers of many files are predictable and the model could be learning the measurements for these headers.
3. The above also means that it is easy for ransomware to bypass any detection strategy that relies on measuring the extremities of the file trivially by not encrypting the extremities.
4. Kurtosis and skew, by themselves, have only a very small positive effect. However, later, we will see that combined with other features, they can have a powerful effect.

5.4 Sprint 2: Effect of Advanced Statistics and Fourier Power Spectral Density

The goal of this sprint was to evaluate the effects of using advanced statistics and Fourier power spectral density on detection accuracy.

5.4.1 Methodology and Implementation

I used the following statistics for the Baseline:

1. Auto-correlation
2. Chi-square statistic
3. Monte Carlo simulation of π
4. Shannon entropy

Pont et al. [33] suggest that kurtosis and skewness can be used to provide better separation between plain-text and ransomware. I therefore used kurtosis and skewness. I also used higher central moments to see if they can also lead to better detection.

Traditionally, Shannon's entropy has been used for ransomware detection. Rényi's entropy has also been used in some cases for better detection. I decided to experiment with several different general entropies including Rényi's entropy, Tsallis' entropy, and Extropy.

It must be pointed out that calculating these entropies and central moments is computationally very efficient; once the byte frequencies have been calculated, computation of the various entropy measurements is cheap.

For advanced statistics, I used the following:

1. Shannon's entropy
2. Rényi's entropy ($\alpha \in [0, 10]$)
3. Tsallis' entropy ($q \in [0, 10]$)
4. Extropy
5. Kurtosis
6. Skew
7. Central moments 2 through 10 (not adjusted for degrees of freedom)

My original hypothesis was that encrypted content creates a more noisy Fourier power spectral density, and autocorrelation, variance, and standard deviation should be able to capture the noisy nature of the spectral density. I also captured the mean because if the mean is high, it would mean that many frequencies have a high power associated with it, and that could also be indicative of encrypted content. I also took higher order moments for experimentation. For a more detailed discussion, please refer to Subsection 4.5.5. I measured the following parameters for this set of features:

1. Autocorrelation
2. Chi-square statistic
3. Mean
4. Statistical moments 2 through 10 (not adjusted for degrees of freedom)
5. Standard deviation

In Section 5.2, I described how I created multiple datasets from the original set of files. I ran various combinations of those datasets, which I controlled by the following boolean filters:

1. Exclude or include plain-text non-base-32.
2. Exclude or include plain-text base-32.

3. Exclude or include v1 encrypted files.
4. Exclude or include v2 encrypted files.
5. Exclude or include v3 encrypted files.
6. Exclude or include encrypted base-32 (v1, v2 or v3).
7. Exclude or include encrypted non-base-32 (v1, v2 or v3).
8. Exclude or include webp (encrypted or plain-text).
9. Exclude or include non-webp (encrypted or plain-text).

The v1, v2 and v3 encryption schemes are described in Section 5.2.

Combining these, I get 54 combinations of datasets, and I ran logistic regression and random forest over these. For each combination, I used 5-fold cross validation. I aggregate and report the mean accuracy, precision, recall and F1 scores over all 54 runs, with special consideration to some cases. I also report the standard deviation in each case.

To show that that beyond a threshold, the number of bytes at the beginning and end that are left unchanged in v2 encryption scheme is inconsequential, I present one set of results with the v3 encryption scheme and show that similar results are produced.

I do not attempt to tune the hyper-parameters on any of the models, since the goal of this project was to test which set of features gives the best performance rather than to find a model that performs best.

5.4.2 Results and Evaluation with Logistic Regression

Table 5.3 compares the mean accuracy, precision, recall, and F1 score the entire dataset when using logistic regression with the different combinations of features. Table 5.4 compares the standard deviations of the same measures. It is observed that the baseline performs quite poorly, and the combination of baseline, advanced, and Fourier statistics performs the best. An improvement in F1 score of 0.19 is seen. In addition, from the table of standard deviations, it is also observed that the combination of baseline, advanced, and Fourier statistics is more consistent across different types of input and encrypted content. This is indicated by the lower standard deviation in Table 5.4.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.783094**	**0.576328**	**0.645605**	**0.586679**
Advanced only	0.837623	0.701169	0.783690	0.678191
Fourier only	0.819778	0.620706	0.641924	0.631164
Baseline and advanced	0.840260	0.710068	0.795446	0.688801
Baseline and Fourier	0.836595	0.685362	0.771181	0.686876
Advanced and Fourier	0.864613	0.760004	0.815969	0.747538
Baseline, advanced and Fourier	**0.867720**	**0.767353**	**0.819309**	**0.755514**

TABLE 5.3: Mean accuracy, F1 score, precision, and recall for the entire data-set with logistic regression

MeasureName Feature	Accuracy	F1	Precision	Recall
Baseline	**0.127518**	0.330339	0.266841	0.370976
Advanced only	0.105039	0.248930	0.171614	0.289986
Fourier only	0.124835	**0.356552**	**0.339809**	**0.372539**
Baseline and Advanced	0.104003	0.242469	**0.134500**	0.282353
Baseline and Fourier	0.117846	0.293607	0.188764	0.330031
Advanced and Fourier	0.103747	0.225121	0.143486	0.266435
Baseline, advanced, and Fourier	**0.102889**	**0.219941**	0.142563	**0.261127**

TABLE 5.4: Standard deviation of accuracy, F1 score, precision and recall for the entire data-set with logistic regression

I devote special attention to webp files. Tables 5.5 and 5.6 compare the mean and standard deviations for webp files alone. Here, a very large improvement in performance is observed with the combination of all three feature sets (F1 improves by 0.18). The performance is also more consistent across all classes of datasets as seen in the table of standard deviations.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.753391	0.473582	0.703834	0.428755
Advanced only	0.802004	0.597768	0.757604	0.539254
Fourier only	**0.749933**	**0.416654**	**0.473101**	**0.415653**
Baseline and Advanced	0.806900	0.615291	0.793781	0.558735
Advanced and Fourier	0.806053	0.634191	0.783672	0.575386
Baseline and Fourier	0.769522	0.523435	0.745747	0.485013
Baseline, advanced, and Fourier	**0.809313**	**0.646285**	**0.784009**	**0.588864**

TABLE 5.5: Mean accuracy, F1 score, precision, and recall for the webp files alone with logistic regression

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.127518**	0.330339	0.266841	0.370976
Advanced only	0.105039	0.248930	0.171614	0.289986
Fourier only	0.124835	**0.356552**	**0.339809**	**0.372539**
Baseline and Advanced	0.104003	0.242469	0.134500	0.282353
Baseline and Fourier	0.117846	0.293607	0.188764	0.330031
Advanced and Fourier	0.103747	0.225121	0.143486	0.266435
Baseline, advanced, and Fourier	**0.102889**	**0.219941**	**0.142563**	**0.261127**

TABLE 5.6: Standard Deviation of accuracy, F1 score, precision, and recall for the webp files only with logistic regression

Finally, for a complete overview, I compare all files other than webp files in Table 5.7. In this case, the biggest improvement is in the F1 score by approximately 0.23.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.815665**	**0.665408**	**0.648940**	**0.698533**
Advanced only	0.891433	0.813084	0.848058	0.797161
Fourier only	0.901990	0.817669	0.798688	0.840048
Baseline and Advanced	0.891002	0.814497	0.844651	0.799063
Baseline and Fourier	0.913370	0.841170	0.848741	0.870257
Advanced and Fourier	0.935861	0.890755	0.890087	0.894823
Baseline, advanced, and Fourier	**0.937962**	**0.893870**	**0.893594**	**0.897522**

TABLE 5.7: Mean accuracy, F1 score, precision, and recall for the non-webp files alone with logistic regression

I conclude this section by with a summary of my findings. Using the combination of all three feature sets—baseline, advanced, and Fourier—gives the best performance. Individually considered, the advanced statistics perform better than Fourier, and the combination of the two performs much better than the baseline. Using a combination of all three feature sets gives more consistent performance across various combinations of input data and encryption schemes.

In the v2 encryption scheme, I did not encrypt the first 128 bytes of the file. To show that the number of bytes left unencrypted does not have much effect, I performed the same experiment with 56 bytes left unencrypted. The average results for all the use-cases is presented in Table 5.8.

The numbers for the v2 encryption scheme presented in Table 5.3 are also similar. This suggests that as long as some bytes in the extremities of the files are left unencrypted, the size of the unencrypted extremities has little effect on the accuracy of detection.

The last observation that I make that is common to all the runs is that the combination of advanced and Fourier features is almost as good as the combination of baseline,

ScoreType	Accuracy	F1	Precision	Recall
Baseline	**0.782983**	**0.575639**	**0.632262**	**0.599188**
Advanced only	0.835497	0.690579	0.782579	0.673004
Fourier only	0.820463	0.621892	0.646605	0.632000
Baseline and advanced	0.837670	0.698041	0.793993	0.682270
Baseline and Fourier	0.833900	0.674746	0.754836	0.680397
Advanced and Fourier	0.861494	0.744864	0.813171	0.735811
Baseline, advanced, and Fourier	**0.864373**	**0.751695**	**0.816463**	**0.743320**

TABLE 5.8: V3 encryption scheme: mean accuracy, F1 score, precision, and recall for the entire data-set with logistic regression

advanced, and Fourier features. Therefore, I can see that the baseline features have very little effect.

In all the above runs, it is observed that the combination of advanced and Fourier features is almost as good as the combination of baseline, advanced, and Fourier features. The baseline, therefore, has very little effect.

5.4.3 Results and Evaluation with Random Forest

I now present the results with a more sophisticated classifier as this is more likely to be used in practice. Table 5.9 captures how the different feature sets compare with different combinations of input plain-text data and encryption schemes.

With random forest, the baseline performs reasonably well. Advanced statistics, when used in isolation, perform better than baseline. Interestingly, Fourier statistics in isolation perform significantly worse than the baseline, but when Fourier statistics are combined with baseline and advanced, the F1 score improves by 0.055.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.930481	0.880013	0.939605	0.835550
Advanced only	0.944956	0.907818	0.955664	0.869634
Fourier only	**0.878111**	**0.734340**	**0.874365**	**0.714305**
Baseline and Advanced	0.952499	0.920670	0.963147	0.886634
Baseline and Fourier	0.939137	0.888630	0.947858	0.853430
Advanced and Fourier	0.959103	0.930944	0.967481	0.901837
Baseline, advanced and Fourier	**0.961236**	**0.934905**	**0.969696**	**0.907093**

TABLE 5.9: Mean accuracy, F1 score, precision, and recall for the data-set with random forest

I next consider webp files alone, and observe the same trend as the entire data-set. In this case, the combination of Fourier, baseline and advanced, statistics yields an

significant increase of 0.045 over the combination of baseline and advanced, and 0.0847 over baseline alone.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.881184	0.786780	0.896356	0.711490
Advanced only	0.915112	0.853293	0.934349	0.792184
Fourier only	**0.810274**	**0.543472**	**0.810364**	**0.504986**
Baseline and advanced	0.919927	0.862616	0.940811	0.803261
Baseline and Fourier	0.883233	0.777226	0.915626	0.697266
Advanced and Fourier	0.922660	0.867648	0.943598	0.809628
Baseline, advanced and Fourier	**0.924665**	**0.871544**	**0.946117**	**0.814310**

TABLE 5.10: Mean accuracy, F1 score, precision, and recall for webp files alone with random forest.

In Table 5.11, I consider non-webp files only. The same trend is observed again, where the combination of baseline, advanced, and Fourier features improves the F1 score by 0.018 over the combination of baseline and advanced statistics.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.972556	0.956461	0.981187	0.934501
Advanced only	0.971724	0.955465	0.979503	0.933875
Fourier only	**0.958172**	**0.919927**	**0.955918**	**0.901808**
Baseline and advanced	0.981003	0.970149	0.986928	0.954709
Baseline and Fourier	0.990470	0.984122	0.990169	0.978490
Advanced and Fourier	0.991132	0.985029	0.992465	0.978058
Baseline, advanced and Fourier	**0.992995**	**0.988343**	**0.993957**	**0.982983**

TABLE 5.11: Mean accuracy, F1 score, precision, and recall for non-webp files alone with random forest

Table 5.12 lists the standard deviations of different metrics for the entire data-set, and Table 5.13 lists the same for webp files only. While using the Fourier statistics alone makes the classification more inconsistent than the baseline, using the combination of all three measurements—baseline, advanced and Fourier statistics—yields the most consistent and robust classification.

From the above tables, it can also be concluded that when using random forests, using Fourier statistics alone may not yield any benefits, but when combined with baseline and advanced statistics, it gives a significant improvement in performance. The improvement in performance for webp files is significant, and for non-webp files is very dramatic. From the standard deviation, it can be seen that combining all features results in a more consistent classification.

It also follows that the combination of Fourier and advanced features is very close in performance to the combination of baseline, advanced, and Fourier features.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.057713	0.114956	0.054548	0.155797
Advanced only	0.043531	0.084977	0.036919	0.120758
Fourier only	**0.098882**	**0.298946**	**0.125534**	**0.337490**
Baseline and advanced	0.040893	0.078630	0.032483	0.114127
Baseline and Fourier	0.060136	0.131571	0.054135	0.182847
Advanced and Fourier	0.040863	0.077458	0.032730	0.112939
Baseline, advanced, and Fourier	**0.039764**	**0.074468**	**0.030772**	**0.109578**

TABLE 5.12: Standard deviation of accuracy, F1 score, precision, and recall for the entire data-set random forest

Feature Set	Accuracy	F1	Precision	Recall
Baseline	0.062108	0.132695	0.062570	0.176662
Advanced only	0.048446	0.100582	0.042906	0.140251
Fourier only	**0.102750**	**0.359471**	**0.136768**	**0.385806**
Baseline and advanced	0.045279	0.092976	0.037155	0.133081
Baseline and Fourier	0.061962	0.156161	0.052769	0.210129
Advanced and Fourier	0.044377	0.090569	0.036402	0.130619
Baseline, advanced, and Fourier	**0.043113**	**0.087194**	**0.034476**	**0.127242**

TABLE 5.13: Standard Deviation of accuracy, F1 score, precision, and recall for the webp files only with random forest.

A look at the *feature_importances_* metric of Scikit Learn's random forest classifier in Figure 5.10 completes the analysis. The feature importance is measured using the mean reduction in impurity in all the random trees as each feature is used to split the data. The mean of the importance of each feature across the different combinations of data-sets is taken. The autocorrelation and standard deviation of the Fourier power spectral density are the most important features. Rényi's entropy, which is very similar to the most probable value, is also very important. The central moments (including skewness) play a very important role.

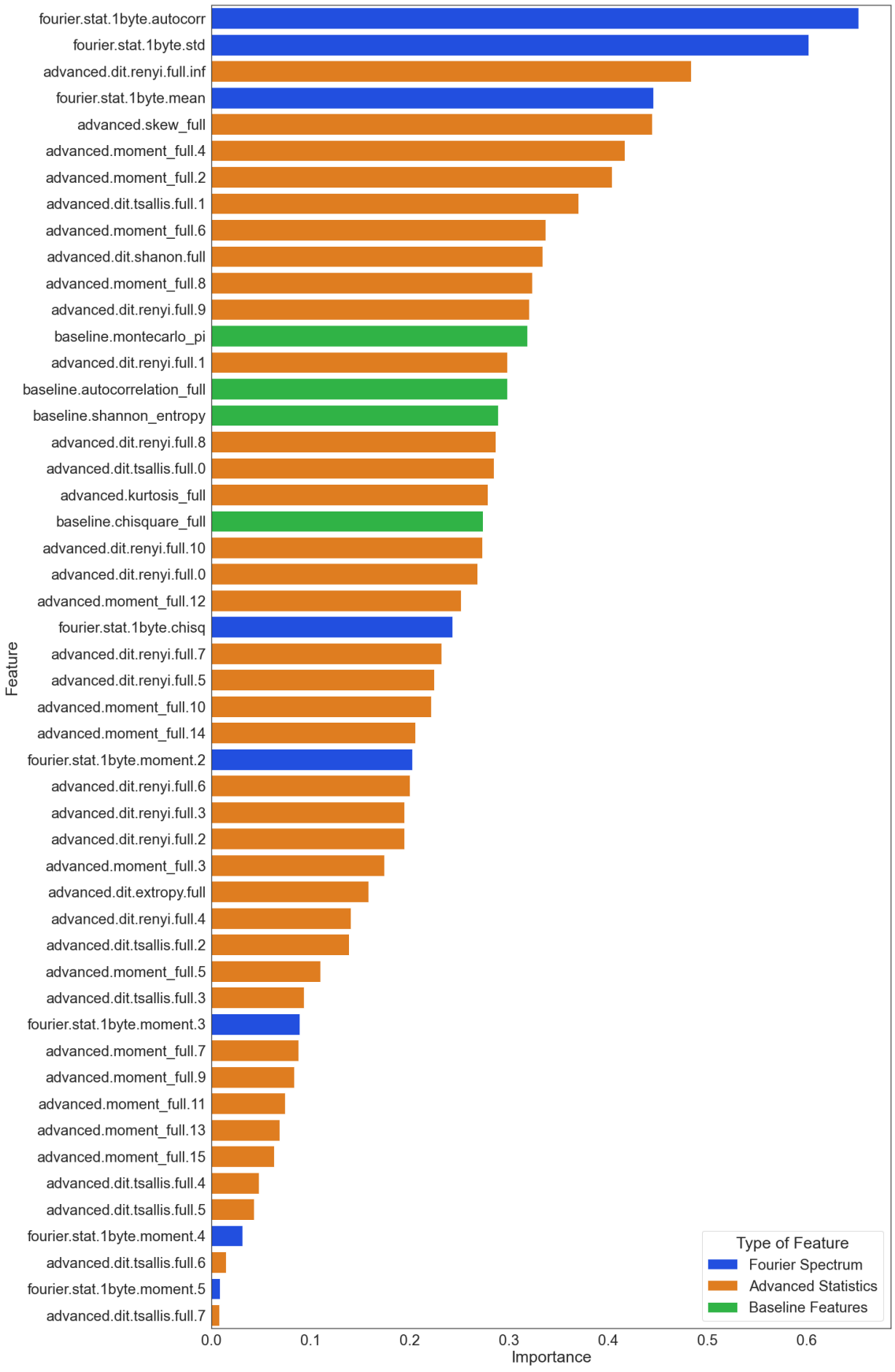

FIGURE 5.10: Mean feature importance for random forest classifier

5.4.4 Review of Key Takeaways

The key findings are listed below.

1. For the logistic regression classifier:
   (a) An improvement of 0.23 in F1 score for non-webp files with the combination of all features is observed.
   (b) An improvement of 0.18 for webp files for the same is seen.
   (c) A combined improvement of 0.19 in the F1 score is seen.
2. For the random forest classifier:
   (a) An improvement of 0.085 in F1 score for non-webp files with the combination of all features is seen.
   (b) An improvement of 0.032 for webp files for the same is observed.
   (c) A combined improvement of 0.055 in F1 score is observed.
3. In both the above cases, when I look at the standard deviation, the combination of all feature sets produces the most consistent detection.
4. The combination of Fourier and advanced features performs almost as well as the combination of all features.
5. The relative importance of all the features is described in Figure 5.10.

5.5 Sprint 3: Effect of a different encryption algorithm

Only AES-128 encryption was used in earlier sprints. In this sprint, I tried to evaluate if changing the encryption algorithm makes a difference. I ran the same experiment as in Section 5.4, except that I used 3DES encryption.

5.5.1 Methodology and Implementation

The design and implementation was very similar to Section 5.4.

5.5.2 Results and Evaluation

For brevity, I do not repeat all the tables I presented in Section 5.4. I only present the average performance.

Table 5.14 presents the results with logistic regression. The combination of all features improves the F1 score by .225 over the baseline.

MeasureName Feature	Accuracy	F1	Precision	Recall
Baseline	**0.833384**	**0.565491**	0.669154	**0.573262**
Advanced only	0.883549	0.747656	0.785304	0.744651
Fourier only	0.843263	0.588228	**0.620431**	0.589351
Baseline and advanced	0.889773	0.764719	0.830693	0.764500
Baseline and Fourier	0.857964	0.666723	0.766824	0.665729
Advanced and Fourier	0.893699	0.777817	0.836229	0.773731
Baseline, advanced and Fourier	**0.897840**	**0.790602**	**0.838211**	**0.789240**

TABLE 5.14: 3DES: Logistic regression

Table 5.15 presents the same results for random forest classifier. The F1 score improves by 0.056.

MeasureName Feature	Accuracy	F1	Precision	Recall
Baseline	0.946530	0.894878	0.941247	0.861136
Advanced only	0.967194	0.938146	0.962383	0.917571
Fourier only	**0.894597**	**0.747369**	**0.861877**	**0.727686**
Baseline and advanced	0.970814	0.945543	0.965722	0.928289
Baseline and Fourier	0.944349	0.888785	0.936755	0.858528
Advanced and Fourier	0.972396	0.948065	0.967240	0.931679
Baseline, advanced and Fourier	**0.973536**	**0.950505**	**0.968267**	**0.935209**

TABLE 5.15: 3DES: Random Forest

Again, it is observed that the combination of Fourier and advanced features comes very close to the same accuracy as the combination of all features.

5.5.3 Review of Key Takeaways

The key findings are summarized below.

1. For logistic regression, there is an improvement of 0.225 in F1 score for non-webp files with the combination of all features.
2. For random forest, an improvement of 0.056 in F1 score for non-webp files with the combination of all features is observed.

3. The performance of the combination of Fourier and advanced features is very close to the performance of the combination of all features.

5.6 Sprint 4: Unseen File Types, Archives, and Password Protected Files

One of the goals of this project is to find features that can be used to recognize unseen file types without the need for retraining. In this Sprint, I trained with the GovDocs1 dataset, and tested with the Napier One dataset. The GovDocs1 dataset is an older dataset, and the types of files in it are limited. The Napier One dataset is a much newer dataset, and has a greater diversity of files. Importantly, this new dataset has the Microsoft Office format files, OpenOffice format files, common archive files, and password protected files. Testing with Napier One was therefore be a good test for how these features perform with file types that weren't seen earlier.

Also, distinguishing between password protected files and files encrypted by ransomware is both necessary and challenging. If the distinction is not made, there will be too many false positives. However, password protection also relies on encryption, which makes the distinction challenging.

Furthermore, distinguishing between encrypted and compressed files is also difficult because both these file types have high Lempel-Ziv complexity.

Existing solutions try to make these distinctions by measuring extremities of files separately [35]. This relies on the fact that legitimate file types have very predictable headers. However, this approach suffers from two drawbacks:

1. If the headers change, the model will not be able to correctly classify these files. New file formats, and updates to existing file formats may need retraining.
2. As we have noted earlier, extremity measurement makes it easier for ransomware to bypass detection.

5.6.1 Methodology and Implementation

I trained two models—logistic regression and random forest—on the GovDocs1 dataset. I then tested with the Napier One dataset and reported the performance on the following parameters:

1. When all files are taken into consideration

2. When dealing with Microsoft office and OpenOffice files; this includes password protected office files
3. With archives; this includes password protected archives
4. With password protected files (office files or archives)

In the context of the discussion in Section 5.6, password protected files are treated as plain-text rather than encrypted.

5.6.2 Results and Evaluation

5.6.2.1 Logistic Regression

Tables 5.16, 5.17, 5.18, and 5.19 capture the performance of the models trained on GovDoc1 dataset when tested against the Napier One dataset for various types of files, namely: all files, office files, archives, and password protected files.

The baseline performance for these file types is very poor and is as good as a coin toss. The advanced measurements do not improve the performance significantly. In fact, when it comes to archives, the predictions from advanced features is worse than a coin toss.

We see that inclusion of Fourier measurements gives a good increase in accuracy (between 0.047 and 0.247). Except in the case of archives, the combination of baseline, Fourier, and advanced features gives the best F1 score. In the case of archives, the Fourier measurements alone give the best F1 score, and the inclusion of either baseline or advanced measurements seems to make the prediction worse.

To remedy this, practical systems that use these measures may use an ensemble of both Fourier only and Fourier with other measurements.

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	0.519106	**0.532106**	0.640025	0.455329
Advanced only	**0.493535**	0.557053	**0.586650**	**0.530299**
Baseline and advanced	0.520351	0.581132	0.610999	0.554049
Fourier only	0.615216	0.673676	0.686448	0.661370
Baseline and Fourier	0.627635	0.676716	**0.706954**	0.648959
Advanced and Fourier	0.626480	0.681560	0.698298	0.665606
Baseline, advanced, and Fourier	**0.636444**	**0.694262**	0.701325	**0.687340**

TABLE 5.16: NapierOne: Results when trained with GovDocs1 and tested on the entire NapierOne dataset with logistic regression

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	**0.499836**	0.547933	0.606428	0.499730
Advanced only	0.364016	**0.234586**	**0.434417**	**0.160676**
Fourier only	**0.549754**	**0.634652**	**0.624885**	**0.644730**
Baseline and Fourier	0.543115	0.622511	0.623948	0.621081
Advanced and Fourier	0.506148	0.563374	0.607439	0.525270
Baseline and advanced	0.404098	0.335709	0.518341	0.248243
Baseline, advanced, and Fourier	0.525820	0.595143	0.617216	0.574595

TABLE 5.19: NapierOne: All archives with logistic regression

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	**0.525803**	**0.522560**	**0.659971**	**0.432508**
Advanced only	0.533491	0.645654	0.593151	0.708355
Baseline and advanced	0.562401	0.666774	0.613851	0.729682
Fourier only	0.646130	0.702760	0.708400	0.697209
Baseline and Fourier	0.693681	0.726220	**0.783010**	0.677111
Advanced and Fourier	0.708689	0.763388	0.744535	0.783219
Baseline, advanced, and Fourier	**0.712954**	**0.769601**	0.742275	**0.799017**

TABLE 5.17: NapierOne: Office Files, including password protected files, with logistic regression

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	0.535857	**0.563248**	0.646805	**0.498810**
Advanced only	**0.426714**	0.585904	**0.517028**	0.675952
Fourier only	0.620143	0.697187	0.668195	0.728810
Baseline and Fourier	**0.664571**	0.713379	**0.731964**	0.695714
Advanced and Fourier	0.635429	0.720910	0.666667	0.784762
Baseline and advanced	0.457143	0.609536	0.536153	0.706190
Baseline, advanced, and Fourier	0.636429	**0.728301**	0.660151	**0.812143**

TABLE 5.18: NapierOne: All password protected files with logistic regression

5.6.2.2 Random Forest

Tables 5.20, 5.21, 5.22, and 5.23 capture the same information as Subsection 5.6.2.1 for random forest.

The predictions using random forest are much better than the predictions using logistic regression. The same trends are seen again. Both the baseline and advanced measurements perform poorly, in some cases no better than a coin toss. The same trends can be observed with the Fourier features (the improvement in F1 score is between 0.039 and 0.238). The Fourier features alone perform the best with password protected archives, and in all other cases, the combination of Fourier with baseline features perform the best. Incorporating advanced features has a detrimental effect in accuracy in most cases.

Given this observation, again, we feel that ensembles of all the combinations must be used.

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	**0.642365**	**0.691574**	**0.717266**	**0.667659**
Advanced only	0.643054	0.708007	0.695847	0.720599
Fourier only	0.738819	0.798984	0.742830	**0.864322**
Baseline and advanced	0.664582	0.722215	0.718414	0.726055
Baseline and Fourier	**0.793450**	**0.829971**	**0.820710**	0.839444
Advanced and Fourier	0.755204	0.791662	0.809642	0.774463
Baseline, advanced, and Fourier	0.761969	0.796238	0.819315	0.774426

TABLE 5.20: Results when trained with GovDocs1 and tested on the entire NapierOne dataset with random forest

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	0.655661	0.706046	0.723712	**0.689222**
Advanced only	**0.625066**	**0.696453**	**0.677168**	0.716869
Fourier only	0.814271	0.853670	0.809505	**0.902931**
Baseline and advanced	0.645129	0.718610	0.685384	0.755222
Baseline and Fourier	**0.886519**	**0.903679**	**0.920758**	0.887221
Advanced and Fourier	0.847709	0.865788	0.918653	0.818676
Baseline, advanced, and Fourier	0.858083	0.876404	0.917779	0.838599

TABLE 5.21: NapierOne: Office files, including password protected files, with random forest

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	**0.510492**	**0.610945**	**0.589811**	**0.633649**
Advanced only	0.547377	0.639697	0.618471	0.662432
Fourier only	**0.611148**	**0.733363**	**0.627791**	**0.881622**
Baseline and Fourier	0.588279	0.709132	0.620428	0.827432
Baseline and advanced	0.526475	0.623231	0.602294	0.645676
Advanced and Fourier	0.579344	0.674407	0.635613	0.718243
Baseline, advanced, and Fourier	0.553033	0.649843	0.619112	0.683784

TABLE 5.22: NapierOne: All archives with random forest, including password protected archives

Feature Set	Accuracy	F1	Precision	Recall
Baseline only	0.495143	0.609848	0.568547	0.657619
Advanced only	0.494143	**0.604225**	0.569412	**0.643571**
Fourier only	0.762857	0.822041	0.747660	**0.912857**
Baseline and Fourier	**0.860571**	**0.886194**	0.868373	0.904762
Advanced and Fourier	0.791714	0.816788	0.864822	0.773810
Baseline and advanced	**0.484714**	0.623211	**0.555183**	0.710238
Baseline, advanced, and Fourier	0.822429	0.847616	**0.873642**	0.823095

TABLE 5.23: NapierOne: All password protected files with random forest

5.6.3 Discussion

This section discusses why Fourier measurements are so effective in dealing with these files when other measurements perform so badly. The chief difficulty here is that password protected files and compressed files are very similar to encrypted files in terms of Lempel-Ziv complexity. Most measures that rely on such complexity will not be able to distinguish between them easily, and hence, existing research has focused on using extremity measurements to distinguish between them. The extremity of a password protected zip file, for example, may have a characteristic signature which can be used to distinguish between them.

Both password protected files and compressed files have sub-structures within the file. The entire file is usually not encrypted or compressed in one go; rather, it is encrypted or compressed in blocks, and each block may have a header. This creates a repeated pattern which is then picked up by Fourier transforms and shows up as a peak.

For example, the *bzip2* format compresses files in blocks which are 900 kilobytes long by default. Each block is independently handled, starts with a 48-bit pattern. These patterns form a repeated pattern, and can be recognized by Fourier transforms.

Metrics like entropy or chi-square are agnostic to any patterns in the files. Autocorrelation may recognize some patterns which are separated by short distances, but patterns separated by long distances cannot be captured by autocorrelation. Fourier transforms can recognize any pattern in the data if it exists, and they will show up as a peak in the frequency spectrum.

5.6.4 Review of Key Takeaways

We summarize the key takeaways.

1. Fourier transforms can be used to classify unseen data types.
2. Fourier transforms are effective in distinguishing between password protected files and encrypted files.
3. Fourier transforms are effective in distinguishing between compressed and encrypted files.
4. Advanced statistics often have a negative effect when password protected or compressed files are involved.
5. In the case of password protected and compressed files, it is not always clear whether Fourier transforms on their own will be more accurate, or when combined with baseline. It may be helpful to use an ensemble of multiple classifiers.

5.7 Sprint 5: Autoencoders

In this sprint, I used autoencoders to evaluate if they can be used to distinguish between plain-text and encrypted content. The rationale here was to train an autoencoder on one class of samples only, and teach it how to reproduce it accurately. This should ideally result in an autoencoder that learns to reconstruct its input with a low reconstruction loss. Now, when it is presented with the same class it is trained on, it should produce a low reconstruction error, and when it is presented a different class from which it is trained on, it should produce a high reconstruction error.

5.7.1 Methodology and Implementation

I trained two types of autoencoders:

1. Densely connected neural network.
2. 1-dimensional convolutional neural network.

For each autoencoder, I used dropout regularization.

I used two different approaches:

1. Train the autoencoder with encrypted data, and use it for prediction.
2. Train the autoencoder with plain-text, and use it for prediction.

5.7.2 Results and Evaluation

The autoencoder network outputs a real number between 0 and 1. To convert this real number to a prediction, a suitable cut-off needs to be determined. The ROC curve is a plot between the true positive rate and the false positive rate as this cut-off is varied. The area under the ROC curve is called the AUC, and ideally the AUC should be high. In this sprint, we report the AUC instead of the accuracy and F1 score.

Table 5.24 shows the AUC scores for each combination of features.

	Trained with Encrypted	Trained with Plaintext
Feature Set	AUC	AUC
Dense: Baseline	0.318603	**0.399032**
Dense: Fourier only	0.395009	0.429468
Dense: Advanced only	0.333478	0.353758
Dense: Baseline and advanced	0.338695	0.452043
Dense: Baseline and Fourier	0.309629	0.368721
Dense: Advanced and Fourier	**0.289166**	0.426582
Dense: Baseline, advanced and Fourier	0.291550	0.390681
Dense: All columns (with power spectrum values)	0.266333	**0.636676**
Dense: Power spectrum values only (1 byte)	0.425826	0.536773
Dense: Power spectrum values only (4 bytes)	**0.448451**	0.502928
1D CNN: Power spectrum values (1 and 4 bytes)	0.430331	0.514078

TABLE 5.24: Autoencoders

As observed, when the autoencoder is trained with encrypted data, it is not able to learn the characteristics of the encoded data well at all. When it is trained with plain-text data, the autoencoder is able to learn some of features of the plain-text data. However, the plain-text data is still too complex for the autoencoder to reproduce. The overall result is that the AUC scores are too low in both cases, and cannot be used for effective detection.

There might still be scope for more work on this by using ensembles of autoencoders.

Also, in this experiment, the autoencoder tried to learn the entire feature set. The use of CNN based autoencoders which work on a moving window of Fourier spectrum values should also be explored.

5.7.3 Review of Key Takeaways

A poor accuracy of detection using autoencoders is observed (AUC = 0.64).

5.8 Sprint 6: Dense Neural Networks

In previous sprints, I did not use deep neural networks. To complete the treatment of the different feature sets, I performed the same experiments as I did in Section 5.4 with fully-connected deep neural networks.

5.8.1 Methodology and Implementation

The neural network architecture is specified in Table 5.25.

Layer Name	Number of Neurons	Activation Function/Value
Dense-8	8	ReLU
dropout_2	8	0.2
Dense-4	4	ReLU
dropout_3	4	0.2
Dense-2	2	ReLU
Dense-1	1	Sigmoid

TABLE 5.25: Architecture of dense neural network

In Section 5.4, I performed 54 different runs with different combinations of the input data. However, here, I just performed one run over the entire dataset.

I took two measurements:

1. I calculated the AUC before setting a cut-off.
2. I set the cut-off at 0.5 and calculated the accuracy, F1 score, precision, and recall.

5.8.2 Results and Evaluation

The results using GovDocs1 dataset is presented in Table 5.26.

Feature Set	Accuracy	F1	Precision	Recall	AUC
Baseline	**0.645**	0.727	**0.591**	0.945	**0.652**
Advanced only	0.823	0.828	0.805	0.853	**0.927**
Fourier only	0.684	0.744	0.626	0.918	0.748
Baseline and advanced	0.751	**0.669**	**1.000**	**0.502**	0.917
Baseline and Fourier	0.706	0.765	0.638	0.954	0.766
Advanced and Fourier	0.816	0.835	0.757	0.930	0.914
Baseline, advanced, and Fourier	**0.818**	**0.842**	0.743	**0.972**	0.912

TABLE 5.26: Deep neural networks with GovDocs1

The best AUC is with advanced statistics alone, but the combination of all (baseline, advanced, and Fourier) comes quite close to it. The combination of baseline and advanced gives the best precision but the worst recall, which indicates that it has too many false positives. This could be because I chose 0.5 as the cut-off, but it is possible to choose a more appropriate value. The accuracy, F1 score, and recall are best with all three feature sets combined. Despite the small differences, using the combination of all three feature sets would still be the preferred approach. The improvement in F1 score over the baseline is 0.115, and the improvement in AUC is 0.26.

I performed another run, this time combining both the GovDocs1 and the NapierOne datasets, and the results are presented in Table 5.27.

Feature Set	Accuracy	F1	Precision	Recall	AUC
Baseline	**0.645**	**0.729**	**0.598**	0.934	**0.646**
Advanced only	0.821	0.840	0.775	0.915	**0.925**
Fourier only	0.685	0.747	0.633	0.910	0.747
Baseline and advanced	0.813	0.820	0.806	**0.834**	0.910
Baseline and Fourier	0.712	0.774	0.646	0.967	0.768
Advanced and Fourier	0.818	0.848	0.741	0.990	0.909
Baseline, advanced, and Fourier	**0.837**	**0.862**	**0.761**	**0.994**	0.918

TABLE 5.27: Deep neural networks with a combination of GovDocs1 and NapierOne

Napier One has more file types and gives a better representation of real world file types. Therefore, the combination of both datasets is more representative of files in the real world. Here we see a clearer trend emerge. We see that F1 and accuracy scores are highest for the combination of all three feature sets, but the AUC is highest when only the advanced statistics are considered. However, even in this case, the combination of both data sets comes quite close to the best AUC score. Overall, combination of all three scores still arguably the best choice. The improvement in F1 score over the baseline is 0.133 and the iprovement in AUC is 0.272.

5.8.3 Review of Key Takeaways

Some key points are summarized below.

1. There is variability in the best scores.
   (a) At times, the advanced statistics perform the best.
   (b) At other times, the combination of all the three feature sets perform the best.
   (c) Even at times when the advanced statics performs the best, the combination of all three is still very close in performance.
2. Overall, the best approach is still to combine all three different feature sets.
3. For the GovDocs1 dataset improvements are seen in the combination of all three feature sets over the baseline:
   (a) 0.115 in F1 score
   (b) 0.26 in AUC
4. For the combination of Napier One GovDocs1 dataset following improvements are observed in the combination of all three feature sets over the baseline:

(a) 0.133 in F1 score
(b) 0.272 in AUC

5.9 Sprint 7: Actual Ransomware Samples

The NapierOne dataset contains 700 files encrypted with 7 actual ransomware programs (100 from each ransomware). This set is too small to draw any meaningful conclusion, but I still used this to get a feel of how these features perform against actual ransomware.

5.9.1 Methodology and Implementation

I performed four different experiments with both logistic regression and random forest:

1. Training was done with the GovDocs1 dataset, similar to Section 5.4. The trained model was then tested on the ransomware samples from the NapierOne dataset.
2. The NapierOne dataset was used both for training and evaluation. The dataset was not encrypted or base-32 encoded. The ransomware files are treated as encrypted, and all other files as plain-text. Since the dataset was heavily imbalanced with 9091 plain-text files and only 700 encrypted files, I used the following two methods to handle the imbalance.
   (a) Use only a subset of the majority class: I used 1818 majority samples
   (b) Use SVMSMOTE on the minority class
3. I wanted to test how well these features perform when tested with ransomware encrypted files that were not used in training. There are encrypted files from seven different ransomware types in the NapierOne dataset. Training was done with files encrypted by six different ransomware, and then tested on the files encrypted with the remaining ransomware. Only a subset of the majority class was used. This was repeated for all seven ransomware.

5.9.2 Results and Evaluation

When GovDocs1 was used for training using a methodology similar to 5.4, and the trained model was tested on ransomware encrypted files from NapierOne, the observed results are described in Table 5.28.

Logistic Regression		
Feature Set	Accuracy	F1
Baseline only	**0.000**	**0.000**
Advanced only	0.068	0.127
Fourier only	**0.109**	**0.196**
Baseline and advanced	0.061	0.115
Baseline and Fourier	0.054	0.102
Advanced and Fourier	0.041	0.079
Baseline, advanced, and Fourier	0.057	0.107
Random Forest		
Feature Set	Accuracy	F1
Baseline only	0.400	0.572
Advanced only	**0.455**	**0.626**
Fourier only	**0.008**	**0.017**
Baseline and advanced	0.427	0.599
Baseline and Fourier	0.088	0.161
Advanced and Fourier	0.354	0.522
Baseline, advanced, and Fourier	0.378	0.548

TABLE 5.28: Results from actual ransomware files when training data didn't contain any ransomware files

Poor performance is noted, and no consistent pattern is found. When logistic regression is used, Fourier features perform the best, but when random forest is used, Fourier features perform the worst.

The model was then trained with samples from Napier One. I considered all files that were encrypted as ransomware as the encrypted class, and all other files as the plain-text class.

Table 5.29 captures the data when training with a subset of the majority class.

Logistic Regression		
Feature Set	Accuracy	F1
Baseline only	**0.712230**	**0.000000**
Advanced only	0.713429	**0.000000**
Fourier only	0.767386	0.517413
Baseline and Fourier	0.764988	0.521951
Advanced and Fourier	**0.826139**	**0.688172**
Baseline and advanced	0.713429	**0.000000**
Baseline, advanced, and Fourier	0.824940	0.685345
Random Forest		
Feature Set	Accuracy	F1
Baseline only	0.864508	0.762105
Advanced only	0.851319	0.730435
Fourier only	**0.824940**	**0.691983**
Baseline and Fourier	0.869305	0.767591
Advanced and Fourier	**0.877698**	**0.782979**
Baseline and advanced	0.866906	0.763326
Baseline, advanced, and Fourier	0.871703	0.770878

TABLE 5.29: Using NapierOne for both training and evaluation, with a sub-set of data

When logistic regression is used, the F1 score is 0 in all the cases where Fourier features are not used. This indicates that there were no true positives. Further the accuracy numbers indicate that logistic regression classified all samples as plain-text when Fourier features are not used. When Fourier features are used, a respectable accuracy of 0.829 and F1 score of 0.688 were achieved, with a very small dataset of just 2500 files and a very simple algorithm like logistic regression.

The use of random forest improves the accuracy of detection across the board. However, it is seen that the combination of advanced and Fourier performs the best. The F1 score improves by 0.021.

From this, it can be concluded that the Fourier features produce a simpler decision boundary which even simple models can learn, and it doesn't need many samples to do so.

The same experiments were repeated with SVMSMOTE to handle the imbalance, and the results are presented in in Table 5.30.

Logistic Regression		
Feature Set	Accuracy	F1
Baseline only	**0.540507**	**0.207889**
Advanced only	0.741187	0.294861
Fourier only	0.752010	0.312178
Baseline and Fourier	0.756339	0.314783
Advanced and Fourier	0.835807	0.404040
Baseline and advanced	0.752319	0.311264
Baseline, advanced, and Fourier	**0.836116**	**0.409800**
Random Forest		
Feature Set	Accuracy	F1
Baseline only	0.930736	0.605634
Advanced only	**0.927335**	0.578097
Fourier only	0.881262	**0.478261**
Baseline and Fourier	0.924552	0.593333
Advanced and Fourier	**0.936302**	**0.630824**
Baseline and advanced	0.936920	0.626374
Baseline, advanced, and Fourier	0.932900	0.625216

TABLE 5.30: Using NapierOne for both training and evaluation, with SVMSMOTE

Here, it is seen that the overall F1 scores are worse than when using the subset of data. This could be because the synthetic examples SVMSMOTE generated confused the models, rather than improving performance. However, the same pattern that the performance is best with the combination of advanced and Fourier feature sets is still observed.

The last experiment evaluated the performance of trained models when they were presented with files encrypted by ransomware that was not seen during training. Ideally, the set of features that is chosen should be able to detect files encrypted by previously unseen ransomware that were not used during testing. The next experiment tested the efficacy of the different combinations of baseline, advanced and Fourier features on files encrypted by unseen ransomware. The NapierOne dataset contains files encrypted by seven different ransomware. The model was trained with files encrypted by 6 different ransomware, and then tested on the files encrypted with the remaining ransomware. This was repeated for all ransomware types.

Feature Set	*DHARMA*	*MAZE*	*NETWALKER*	*NOTPETYA*	*PHOBOS*	*RYUK*	*SODINOKIBI*
			Accuracy				
Baseline only	**0.667**	**0.667**	**0.667**	**0.667**	**0.667**	**0.667**	**0.667**
Advanced only	0.747	0.797	**0.540**	0.727	0.757	0.807	0.793
Fourier only	0.783	0.850	0.583	0.740	0.780	0.847	0.850
Baseline and Fourier	0.783	0.837	0.573	0.747	0.783	0.837	0.837
Advanced and Fourier	**0.840**	**0.910**	0.600	0.800	**0.843**	**0.913**	**0.903**
Baseline and advanced	0.763	0.843	0.553	0.787	0.787	0.847	0.853
Baseline, advanced, and Fourier	**0.840**	0.907	0.600	**0.807**	**0.843**	0.910	0.900
			F1 Score				
Baseline only	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
Advanced only	0.604	0.690	0.028	0.564	0.626	0.710	0.684
Fourier only	0.683	0.798	**0.126**	0.598	0.676	0.793	0.798
Baseline and Fourier	0.686	0.782	0.123	0.608	0.686	0.782	0.782
Advanced and Fourier	**0.767**	0.880	0.091	0.691	**0.773**	**0.885**	**0.870**
Baseline and advanced	0.628	0.771	0.043	0.677	0.677	0.777	0.788
Baseline, advanced, and Fourier	**0.767**	**0.876**	0.091	**0.701**	**0.773**	0.881	0.866

TABLE 5.31: Unseen ransomware: accuracy and F1 score with logistic regression for ransomware not used in training

Table 5.31 presents the accuracy and F1 scores when this experiment was run with logistic regression.

It is observed that the F1 scores for the baseline are all 0. This is because the baseline could not predict the ransomware class correctly at all. Except for NETWALKER, either the combination of advanced and Fourier features, or the combination of baseline, advanced and Fourier features performs the best. On the whole, the the combination of baseline, advanced and Fourier features is recommended.

Table 5.32 describes the results when the same experiment was performed with random forest.

The results are more mixed in this case. When Fourier features are used in isolation, they perform the worst, but their performance improves when combined with baseline and advanced features. Except for the DHARMA and NETWALKER ransomware, the combination of baseline, advanced and Fourier features improves upon the baseline

features alone. The combination of baseline, advanced and Fourier still performs the best when the average F1 score across all ransomware files is considered.

Feature Set	*DHARMA*	*MAZE*	*NETWALKER*	*NOTPETYA*	*PHOBOS*	*RYUK*	*SODINOKIBI*
			Accuracy				
Baseline only	0.913	0.897	**0.637**	**0.853**	0.917	0.903	0.907
Advanced only	**0.887**	0.897	0.610	0.837	0.923	0.900	0.907
Fourier only	0.890	**0.890**	**0.607**	**0.817**	**0.887**	**0.867**	**0.873**
Baseline and advanced	0.913	0.910	0.630	0.853	0.923	**0.930**	**0.923**
Baseline and Fourier	**0.917**	**0.913**	0.623	0.833	0.920	0.907	0.920
Advanced and Fourier	0.910	**0.913**	0.633	0.843	0.937	0.927	0.920
Baseline, advanced, and Fourier	0.910	**0.913**	0.633	**0.853**	**0.943**	0.923	0.920
			F1 Score				
Baseline only	0.877	0.852	**0.128**	**0.768**	0.884	0.863	0.869
Advanced only	**0.835**	0.853	**0.033**	0.738	0.893	0.856	0.867
Fourier only	0.849	**0.848**	0.063	**0.706**	**0.847**	**0.811**	**0.822**
Baseline and advanced	0.875	0.870	0.098	**0.768**	0.892	**0.900**	**0.892**
Baseline and Fourier	**0.884**	**0.877**	0.066	0.731	0.889	0.867	0.889
Advanced and Fourier	0.868	0.876	0.083	0.746	0.912	0.895	0.885
Baseline, advanced, and Fourier	0.870	0.876	0.083	**0.768**	**0.921**	0.889	0.886

TABLE 5.32: Unseen ransomware: accuracy and F1 score with random forest for ransomware not used in training

The mean values from Table 5.31 and Table 5.32 are averaged and presented in Table 5.33. This table shows that in all cases, inclusion of Fourier features results in better accuracy and F1 scores. The combination of baseline, advanced and Fourier features is preferred.

It can be concluded that the combination of baseline, advanced and Fourier features is more resilient against ransomware which was not seen during training. It must also be noted that the number actual ransomware encrypted samples is very small (just 700), and further testing with more ransomware encrypted samples is needed.

	Logistic regression		Random forest	
Feature Set	Accuracy	F1 score	Accuracy	F1 score
Baseline only	**0.667**	**0.000**	0.861	0.749
Advanced only	0.738	0.558	**0.851**	0.725
Fourier only	0.776	0.639	0.833	**0.707**
Baseline and advanced	0.776	0.623	0.869	**0.756**
Baseline and Fourier	0.771	0.636	0.862	0.743
Advanced and Fourier	**0.830**	**0.708**	0.869	0.752
Baseline, advanced, and Fourier	**0.830**	**0.708**	**0.871**	**0.756**

TABLE 5.33: Unseen ransomware: mean accuracy and F1 scores over all unseen ransomware

5.9.3 Review of Key Takeaways

The key takeaways are summarized below.

1. If ransomware samples are not used in training, the classification accuracy is poor.
2. The use of Fourier analysis can give:
   (a) better accuracies with simple models
   (b) a better accuracy with a small dataset
   (c) a cleaner decision boundary
3. The use of Fourier analysis improves accuracy across different models.
4. The combination of baseline, advanced and Fourier features performs well with ransomware that was not seen during training.

5.10 Sprint 8: Sampling

Speed and resource utilization are important considerations in practice. Sometimes, it may be necessary to make some sacrifice in detection accuracy for the benefit of speed and low resource utilization.

This is especially true of slower disks where reading large files may cause performance problems.

In such cases, sampling the files, instead of reading the entire file, may be an acceptable compromise.

In this sprint, I explored a strategy to sample bytes from a file instead of reading the entire file as a balance between detection accuracy and detection speed.

5.10.1 Methodology and Implementation

For each file, I took 4 random samples of 256 bytes each. I concatenated these samples together, and then calculated the features for each joined sample.

I then calculated the performance of the model for each of the feature sets. I used 5-fold cross validation. I repeated the experiment with both logistic regression and random forest.

5.10.2 Results and Evaluation

5.10.2.1 Sampling with Logistic Regression

Table 5.34 describes the results of running the full dataset with logistic regression. For reference, the accuracy and F1 score for the entire file were 0.705724 and 0.734097 respectively.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.761149**	**0.820384**	**0.712656**	**0.969040**
Advanced only	0.791496	0.838543	0.747503	0.960246
Fourier only	0.799342	0.844455	0.759997	**0.957614**
Baseline and Fourier	0.815050	0.857171	0.773712	0.967551
Baseline and advanced	0.796065	0.841875	0.752103	0.961508
Advanced and Fourier	0.829093	0.866200	0.790879	0.964272
Baseline, advanced, and Fourier	**0.829909**	**0.866911**	**0.791466**	0.965094

TABLE 5.34: Sampling with logistic regression on GovDocs1

The above shows the combination of all three datasets performs the best. Curiously, the sampling strategy actually performs better than reading the complete file.

Table 5.35 describes the result for webp files alone with logistic regression. For reference, the accuracy and F1 score for the entire file were 0.62285 and 0.63599 respectively.

Feature Set Feature	Accuracy	F1	Precision	Recall
Baseline	**0.731670**	**0.807304**	**0.678283**	**0.996952**
Advanced only	0.743690	0.813273	0.690044	**0.990177**
Fourier only	0.736425	0.810271	0.681814	0.998404
Baseline and advanced	0.743937	0.813420	0.690255	0.990185
Baseline and Fourier	0.736167	0.808478	0.683992	0.988365
Advanced and Fourier	0.751828	0.817090	0.698938	0.983409
Baseline, advanced, and Fourier	**0.752324**	**0.817343**	**0.699484**	0.983062

TABLE 5.35: Sampling with logistic regression for webp files

Again, the same trend is observed—the combination of all three feature sets performs the best, and also the sampling strategy performs better than reading the entire file.

Table 5.36 captures the results for non-webp files. For reference, the accuracy and F1 score for the entire file were 0.841066 and 0.840996 respectively.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.788047**	**0.833097**	**0.745485**	0.945247
Advanced only	0.833047	0.859083	0.809128	**0.915920**
Fourier only	0.854915	0.877942	0.833682	0.929257
Baseline and advanced	0.841288	0.865797	0.816824	0.921441
Baseline and Fourier	0.883004	0.901804	0.858772	0.950324
Advanced and Fourier	0.898849	0.912763	0.884084	0.943650
Baseline, advanced, and Fourier	**0.899744**	**0.913616**	**0.884826**	**0.944630**

TABLE 5.36: Sampling with logistic regression for non-webp files

Again, the same trend is observed as the entire dataset—the combination of all three feature sets performs the best, and also the sampling strategy performs better than reading the entire file.

5.10.2.2 Sampling with Random Forest

Table 5.37 captures the results of the entire dataset with Random Forest. For reference, the accuracy and F1 score for the entire file were 0.940441 and 0.939706 respectively.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.841589**	**0.869127**	**0.824341**	**0.921938**
Advanced only	0.856911	0.879295	0.842323	0.921042
Fourier only	0.840270	0.875990	0.794997	**0.980759**
Baseline and Fourier	0.874727	0.900685	0.838610	0.977858
Baseline and advanced	0.875433	0.895071	0.859397	0.935410
Advanced and Fourier	0.889765	0.907584	0.872106	0.947815
Baseline, advanced, and Fourier	**0.896079**	**0.913007**	**0.877431**	0.953400

TABLE 5.37: Sampling with random forest on GovDocs1

The combination of all three feature sets performs the best. Unlike the case with logistic regression, a drop of 0.027 in the F1 score is observed as compared to the accuracy when the whole file was used.

Table 5.38 captures the results for webp files only. For reference, the accuracy and F1 score for the entire file were 0.870261 and 0.8626 respectively.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.752663**	**0.802702**	0.729049	0.893262
Advanced only	0.791105	0.827587	0.775267	**0.888390**
Fourier only	0.754345	0.816955	**0.704129**	**0.972863**
Baseline and advanced	0.803931	0.838344	0.784452	0.901256
Baseline and Fourier	0.774823	0.827347	0.728384	0.957685
Advanced and Fourier	0.808669	0.842223	0.788393	0.905112
Baseline, advanced, and Fourier	**0.814403**	**0.847128**	**0.792114**	0.911589

TABLE 5.38: Sampling with random forest for webp files

Again, the combination of all three feature sets performs the best. Unlike the case with logistic regression, however, a drop of 0.015 is sen in the F1 score.

Table 5.39 illustrates the results for non-webp files only. For reference, the accuracy and F1 score for the entire file were 0.981146 and 0.980983 respectively.

Feature Set	Accuracy	F1	Precision	Recall
Baseline	**0.919635**	0.928421	0.919547	**0.937475**
Advanced only	0.918046	**0.927430**	**0.913521**	0.941828
Fourier only	0.916592	0.930504	0.886024	0.982005
Baseline and advanced	0.940989	0.947623	0.936760	0.958784
Baseline and Fourier	0.964950	0.969387	0.952097	0.987651
Advanced and Fourier	0.963315	0.967619	0.957662	0.977898
Baseline, advanced, and Fourier	**0.970615**	**0.974057**	**0.964669**	**0.983726**

TABLE 5.39: Sampling with random forest for non-webp files

Like in the previous table, the combination of all three feature sets performs the best. Again, a drop of 0.006 in F1 score is observed.

Overall, the drop in the F1 score is very low, and as such, the gains from speed may be worth the sacrifice in accuracy.

5.10.3 Review of Key Takeaways

The key takeaways are listed below.

1. Sampling can lead to faster detection, and less resource utilization.
2. Results from sampling are comparable to results from the entire file.
   (a) In the case of logistic regression, there is a significant improvement in accuracy when sampling is used.
   (b) In the case of random forest, there is a small decrease in accuracy when sampling is used.
3. In all cases, the combination of all three feature sets performs the best.

Chapter 6

Discussion and Conclusions

6.1 Discussion

I set off on this project with the following goals in mind:

1. To define and evaluate a set of features for robust detection of encrypted files across a variety of file types, especially webp files.
2. To systematically study the effectiveness of the following measurements to discriminate between encrypted and plain content:
   (i) Fourier power spectrum density
   (ii) Kurtosis
   (iii) Skewness
   (iv) Central moments
3. To systematically study the effectiveness of the above methods to provide robust defense against anti-detection techniques used by ransomware like:
   (i) Not encrypting the first few bytes of a file
   (ii) Encrypting alternate blocks of n bytes
   (iii) Entropy reduction techniques
4. To differentiate between encrypted content, password protected content and compressed files using the features described earlier. This is necessary for reducing false positives.
5. To use, evaluate, and optimize a combination of the above features with traditionally used statistical measures along with an autoencoder network to provide a robust method to detect encrypted content

6. To improve speed of detection by using a sampling strategy without sacrificing too much accuracy

In Chapter 3, I demonstrated why there is a need for more features that capture the underlying structure of a file. A fully connected deep neural network was trained on a set of 50,000 files, and it achieved an F1 score of 0.947. However, when the trained model was presented with files that were base-32 encoded after encryption, the model performed poorly with an F1 score of 0.0.

In Section 5.3, a baseline for comparison was established with the most commonly used statistics in commercial anti-ransomware solutions. I found that measuring the statistics of the extremities of the files improves accuracy by about 0.10. However, the corollary to this observation is that ransomware can simply not encrypt the extremities of a file to avoid detection.

In Section 5.4, I showed that incorporating advanced statistics and Fourier power spectral density can improve accuracy of detection by a large amount. When advanced and Fourier features were incorporated, the F1 scores for the entire dataset improved by 0.191 with logistic regression and 0.055 with random forest. Further, it was also observed that incorporating the same features improved detection across a variety of file types including webp files. For webp files, the F1 scores improved by 0.173 and 0.085 with logistic regression and random forest respectively. The results also indicated resilience when files were only partially encrypted; with the v3 encryption scheme, the in F1 score improved by 0.176. Lastly, I also observed more consistent detection across file types. The standard deviation of F1 scores reduced by 0.11 and 0.04 for logistic regression and random forest respectively.

In Section 5.5, I demonstrated that incorporating advanced statistics and Fourier power spectral density into the set of features gives the same benefits across different encryption algorithms. For 3DES, an increase in F1 score of 0.225 with logistic regression was observed. With random forest, the F1 score improved by 0.056. These numbers are similar in scale to those for AES-128 encryption described in Section 5.4; therefore, it is reasonable to assume that the combination of baseline, advanced, and Fourier features works well regardless of the encryption algorithm used.

The results from Section 5.6 demonstrated that using a combination of baseline, advanced, and Fourier features improves the classification of unseen file types. The F1 score for unseen file types improves by 0.163 with logistic regression, and by 0.105 with random forest. It was also observed that the use of Fourier power spectral density provides a good decision boundary to distinguish between encrypted files from password protected files, office files, and archives. For password protected files, the improvements

in F1 score for logistic regression and random forest were 0.047 and 0.039 respectively. For office files, the the improvements in the same scores were 0.247 and 0.170, and for password protected files, the improvements were 0.165 and 0.238.

The use of autoencoders was attempted in Section 5.7, but the performance was poor. The best AUC score observed was only 0.637.

Section 5.8 described the use of these features with dense fully-connected neural networks, and the result was an impressive improvement in accuracy with the combination of all three feature sets. When only the GovDocs1 was used, the F1 score improved by 0.115 and the AUC improved by 0.26. When a combination of GovDocs1 and NapierOne was used, the F1 score improved by 0.192 and the AUC by 0.272.

In Section 5.9, the same feature sets were tested against a small collection of files encrypted by actual ransomware programs, and a modest improvement in F1 score and accuracy was observed. The dataset was imbalanced and I used two techniques to balance it. When I balanced it by taking a subset of the majority class, the improvement in F1 score with the combination of baseline, advanced, and Fourier features over the baseline features was 0.685 with logistic regression, and 0.009 with random forest. When I used SMOTE to handle the data imbalance, the improvement in the combination of baseline, advanced and Fourier features over the baseline features was 0.202 with logistic regression and 0.020 with random forest.

In Section 5.9, I also tested the how effective the different combinations of baseline, advanced and Fourier features were when they were used to classify files encrypted by ransomware that were not used during training. I found that, on an average, the combination of baseline, advanced and Fourier features performs the best when classifying files encrypted by previously unseen ransomware.

The effect of using sampling for speed and efficiency of detection was explored in Section 5.10. When logistic regression was used, the use of the combination of baseline, advanced, and Fourier features improved the F1 score over the baseline features by 0.047 for the entire dataset and by 0.010 for webp files. When random forest was used for the same, the combination of baseline, advanced, and Fourier features improved F1 score over the baseline features by 0.044 for both the entire dataset and for webp files. Furthermore, the results showed that in the case of logistic regression, sampling actually improved accuracy. When the files were sampled, the F1 score was 0.867, but the F1 score was only 0.734 when the entire file was used. In the case of random forest, sampling resulted in a small degradation in F1 score. When the entire file was used, the F1 score was 0.940, but when the file was sampled, the F1 score was 0.913.

Overall, improvements in the F1 score ranging from 0.02 to 0.27 were observed.

The observation that Fourier power spectral density can be used to distinguish between encrypted content, password protected files, and compressed files is of significant importance as this is a difficult problem. The approach is promising but must be explored further with focused attention.

With the exception of the use of autoencoders, all the objectives that I set at the start of the project have been achieved. However, there is scope for further improvement. I did not test against a large set of files encrypted with actual ransomware, and given more time, I would have liked to do more work on that.

6.2 Challenges Faced and Resources Utilized

During the course of this project, significant challenges were encountered. The amount of data to be processed was large and I/O speed was always a bottleneck. The large storage requirements posed problems of their own as it became difficult to share the files on Google Drive so that they could be imported on Google Colab.

I also faced challenges in understanding the topic. The courses by Christof Paar on cryptography and Steve Brunton on Fourier analysis were especially helpful in getting a firm hold on the theory to understand the problem space. Several courses on statistics on the internet were also helpful in understanding some nuances of the different statistics I used. Courses on information theory from the Government of India's NPTEL initiative proved to be of much assistance.

6.3 Criticism of Project Execution

When I set out on this project, I had proposed the timelines as described in Figure 6.1.

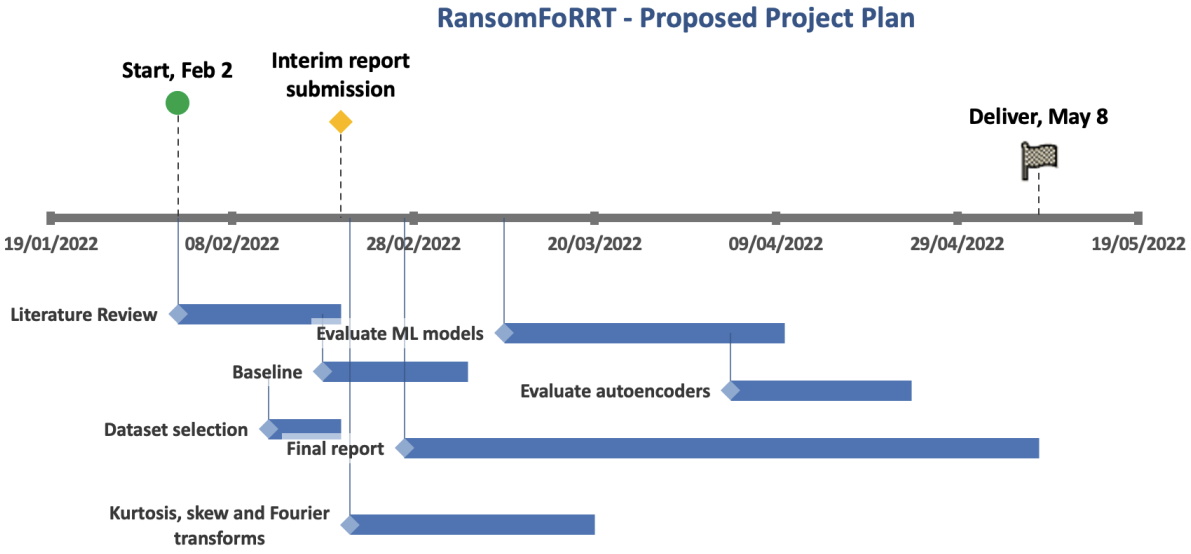

FIGURE 6.1: Proposed Timelines

During the course of the project, I faced several challenges with data preprocessing, and not all data could be preprocessed at the same time. The consequence of this was that some sprints had to be broken down into multiple sprints. For example, the experiments with 3DES were done in a separate sprint.

During the course of this work, I realized how I/O can become a bottleneck and slow down data preprocessing. It inspired me to evaluate the use of sampling to improve speed and efficiency. This increased the scope of the work, and a sprint was added.

The work was also slowed down because of the power requirements of preprocessing the dataset. My laptop would discharge even when connected to A/C power, and every few hours, I had to shut down my laptop and leave it idle to charge back up. This issue was resolved by ordering a new power adapter and cable, but this added unexpected delays in the initial sprints.

Some sprints took longer than I expected them to because of unexpected issues with my day job.

The timelines were adjusted accordingly, and the actual timeline is presented in Figure 6.2.

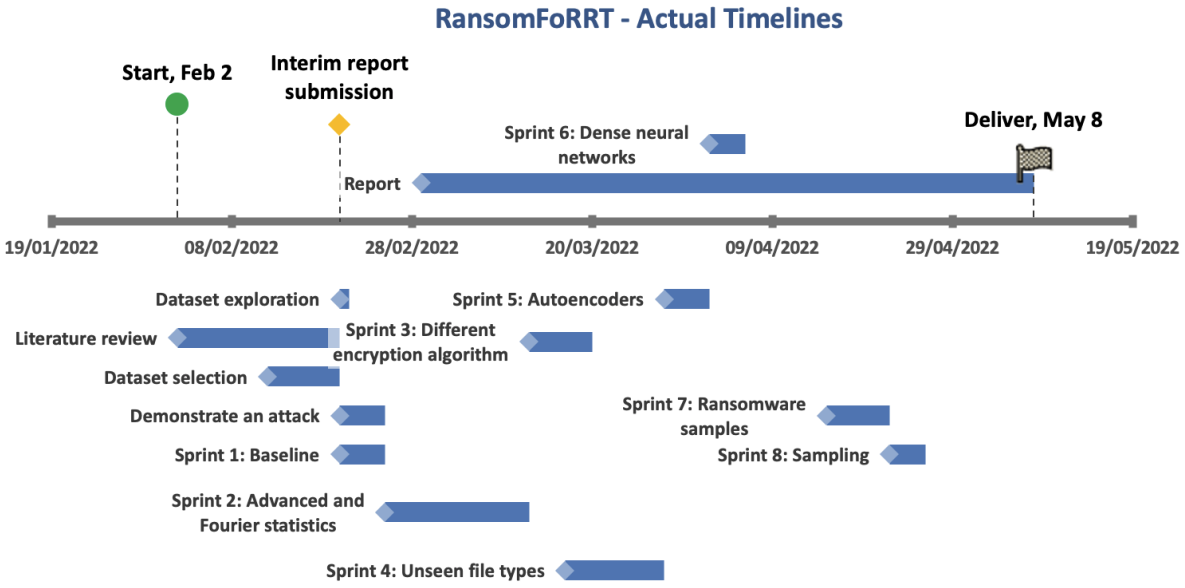

FIGURE 6.2: Actual timelines

6.4 Lessons Learned

During the implementation of this work, I faced considerable challenges with sluggish I/O on slow hard drives that slowed down the implementation substantially. This made me realize that this would be an important consideration for any real world production system.

The second problem I faced was power consumption. While training the models, my laptop was draining battery power even while connected on A/C power. This was because my cable was not able to supply enough power to the laptop when all CPUs were running full throttle. This illustrated the need for reducing the CPU and power consumption.

The above challenges inspired me to think of ways to make this work more useful for real world systems, and I added a sprint on evaluating a sampling strategy in this project.

On another front, I realized the importance of leaving buffers while planning. For one week during this project, I had certain issues at my day job and I wasn't able to concentrate much on the project. Also, some of the sprints took longer than expected due to slow I/O speed. Furthermore, the scope of the project was extended. All of this resulted in extra pressure on me. Leaving some buffers for unexpected hurdles in the planning phase would have been of great help.

6.5 Conclusion

I demonstrated that the combination of baseline statistics, advanced statistics and Fourier features can increase the accuracy of detecting content encrypted by ransomware. I also demonstrated that this is resilient to tweaks that ransomware programs use to avoid detection. I showed that the combination of these features is resilient against unknown file types. Webp files are hard to distinguish from encrypted content; the combination of these three feature sets is able to better distinguish between them. These features can be used along with sampling in production machines where it may be necessary to reduce resource utilization. One of the most interesting results in this work is that Fourier power spectral density is able to distinguish between compressed files, password protected files, and encrypted files.

A paper submission to a conference based on this work is planned.

All of the code written for this work can be accessed from the following GitHub repository:
https://github.com/bhattacharjee/RansomFoRRT.

6.6 Future Work

There are several avenues where this work could be expanded.

1. I only used statistics of the Fourier power spectral density, and did not use the values as such. In doing so, I discard a lot of information, and future work should explore the use of those values in ransomware detection.
2. The combination of different features should be explored. For example, the product of the standard deviation and mean of the Fourier power spectral density may give a good decision boundary.
3. The effect of noise reduction by clipping frequencies whose power is much below the frequencies with the highest power must be explored.
4. In Subsection 5.6.2, it was observed that there might be scope for using an ensemble of different classifiers. Specifically, classifiers that use different feature sets as well as classifiers that use different data sets should be explored. For an example of the latter, different classifiers can be trained on data that either includes base-32 or does not include base-32 encrypted files. Different classifiers can also be trained on different encryption algorithms or tweaks to them. The results from each classifier can be used as an input for a secondary classifier.

5. It was seen that Fourier analysis alone can be used to effectively distinguish between password protected files, encrypted files, and compressed files. This should be explored systematically in more detail.

6. I have used Fourier transforms where I considered 1 byte and 4 bytes at a time. There are several other possibilities which must be investigated:

   (a) The use of wavelet transforms in place of Fourier transforms

   (b) The use of Fourier and wavelet transforms on bits rather than bytes

7. The use of an ensemble of CNN based autoencoders which work on a moving window of Fourier spectrum values should be explored.

Bibliography

[1] T. McIntosh, A. Kayes, Y.-P. P. Chen, A. Ng, and P. Watters, "Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–36, 2021.

[2] S. Sjouwerman, "Council post: Seven factors analyzing ransomware's cost to business," Jul 2021. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/07/29/seven-factors-analyzing-ransomwares-cost-to-business/?sh=72a01ff42e98

[3] T. Anderson and W. Torreggiani, "The impact of the cyberattack on radiology systems in ireland," *Irish Medical Journal*, vol. 114, no. 5, p. 137, 2021.

[4] K. A. W. Ramsdell and K. E. Esbeck, "Evolution of ransomware," 2021.

[5] M. Moran Stritch, M. Winterburn, and F. Houghton, "The conti ransomware attack on healthcare in ireland: Exploring the impacts of a cybersecurity breach from a nursing perspective," *Canadian Journal of Nursing Informatics*, vol. 16, no. 3-4, 2021.

[6] T. Cymru, "Analyzing ransomware negotiations with conti: An in-depth analysis."

[7] PWC Corporation, "Conti Cyber Attack on the HSE." [Online]. Available: https://www.hse.ie/eng/services/publications/conti-cyber-attack-on-the-hse-full-report.pdf

[8] CrowdStrike, "Most common types of ransomware: Crowdstrike," Feb 2022. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/ransomware/types-of-ransomware/

[9] A. H. Mohammad, "Ransomware evolution, growth and recommendation for detection," *Modern Applied Science*, vol. 14, no. 3, p. 68, 2020.

[10] M. Loman, "Lockfile ransomware's box of tricks: Intermittent encryption and evasion," Sep 2021. [Online]. Available: https://news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-of-tricks-intermittent-encryption-and-evasion/

[11] J. Lee and K. Lee, "A method for neutralizing entropy measurement-based ransomware detection technologies using encoding algorithms," *Entropy*, vol. 24, no. 2, p. 239, 2022.

[12] M. Loman, "How ransomware attacks," 2019.

[13] S. Kok, A. Abdullah, N. Jhanjhi, and M. Supramaniam, "Ransomware, threat and detection techniques: A review," *Int. J. Comput. Sci. Netw. Secur*, vol. 19, no. 2, p. 136, 2019.

[14] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Review of current ransomware detection techniques," in *2021 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE, 2021, pp. 1–6.

[15] R. Moussaileb, N. Cuppens, J.-L. Lanet, and H. L. Bouder, "A survey on windows-based ransomware taxonomy and detection mechanisms," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[16] H. Guo, S. Huang, C. Huang, Z. Pan, M. Zhang, and F. Shi, "File entropy signal analysis combined with wavelet decomposition for malware classification," *IEEE Access*, vol. 8, pp. 158 961–158 971, 2020.

[17] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, and A. K. Sangaiah, "Classification of ransomware families with machine learning based onn-gram of opcodes," *Future Generation Computer Systems*, vol. 90, pp. 211–221, 2019.

[18] F. Manavi and A. Hamzeh, "A new method for ransomware detection based on pe header using convolutional neural networks," in *2020 17th International ISC Conference on Information Security and Cryptology (ISCISC)*. IEEE, 2020, pp. 82–87.

[19] Microsoft, "PE format - win32 apps." [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

[20] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 21–30.

[21] C. M. Hartman, "Cross-architecture binary function fingerprinting."

[22] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay, "Ratafia: ransomware analysis using time and frequency informed autoencoders," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 218–227.

[23] M. A. Ayub, A. Continella, and A. Siraj, "An i/o request packet (irp) driven effective ransomware detection scheme using artificial neural network," in *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE, 2020, pp. 319–324.

[24] G. Cusack, O. Michel, and E. Keller, "Machine learning-based detection of ransomware using sdn," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2018, pp. 1–6.

[25] TeamSymantec, "Ransomware: Growing number of attackers using virtual machines." [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/ransomware-virtual-machines

[26] P. Penrose, R. Macfarlane, and W. J. Buchanan, "Approaches to the classification of high entropy file fragments," *Digital Investigation*, vol. 10, no. 4, pp. 372–384, 2013.

[27] K. Lee, S.-Y. Lee, and K. Yim, "Machine learning based file entropy analysis for ransomware detection in backup systems," *IEEE Access*, vol. 7, pp. 110 205–110 215, 2019.

[28] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 303–312.

[29] V. Roussev, "Data fingerprinting with similarity digests," in *IFIP International Conference on Digital Forensics*. Springer, 2010, pp. 207–226.

[30] M. Bat-Erdene, H. Park, H. Li, H. Lee, and M.-S. Choi, "Entropy analysis to classify unknown packing algorithms for malware detection," *International Journal of Information Security*, vol. 16, no. 3, pp. 227–248, 2017.

[31] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Differential area analysis for ransomware attack detection within mixed file datasets," *Computers & Security*, vol. 108, p. 102377, 2021.

[32] S. Jung and Y. Won, "Ransomware detection method based on context-aware entropy analysis," *Soft Computing*, vol. 22, no. 20, pp. 6731–6740, 2018.

[33] J. Pont, B. Arief, and J. Hernandez-Castro, "Why current statistical approaches to ransomware detection fail," in *International Conference on Information Security*. Springer, 2020, pp. 199–216.

[34] T. McIntosh, J. Jang-Jaccard, P. Watters, and T. Susnjak, "The inadequacy of entropy-based ransomware detection," in *International Conference on Neural Information Processing*. Springer, 2019, pp. 181–189.

[35] C.-M. Hsu, C.-C. Yang, H.-H. Cheng, P. E. Setiasabda, and J.-S. Leu, "Enhancing file entropy analysis to improve machine learning detection rate of ransomware," *IEEE Access*, vol. 9, pp. 138 345–138 351, 2021.

[36] N. Bailluet, H. Le Bouder, and D. Lubicz, "Ransomware detection using markov chain models over file headers," in *SECRYPT 2019: 16th International Conference on Security and Cryptography*, 2021.

[37] B. Zhao, Q. Liu, and X. Liu, "Evaluation of encrypted data identification methods based on randomness test," in *2011 IEEE/ACM international conference on green computing and communications*. IEEE, 2011, pp. 200–205.

[38] M. Alam, S. Sinha, S. Bhattacharya, S. Dutta, D. Mukhopadhyay, and A. Chattopadhyay, "Rapper: Ransomware prevention via performance counters," *arXiv preprint arXiv:2004.01712*, 2020.

[39] M. McDaniel and M. H. Heydari, "Content based file type detection algorithms," in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. IEEE, 2003, pp. 10–pp.

[40] W.-J. Li, K. Wang, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE, 2005, pp. 64–71.

[41] Q. Li, A. Ong, P. Suganthan, and V. Thing, "A novel support vector machine approach to high entropy data fragment classification," in *Proceedings of the South African Information Security Multi-Conf (SAISMC)*, 2011, pp. 236–247.

[42] C. J. Veenman, "Statistical disk cluster classification for file carving," in *Third international symposium on information assurance and security*. IEEE, 2007, pp. 393–398.

[43] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Transactions on information theory*, vol. 22, no. 1, pp. 75–81, 1976.

[44] A. N. Kolmogorov, "On tables of random numbers," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 369–376, 1963.

[45] L. Sportiello and S. Zanero, "File block classification by support vector machine," in *2011 Sixth international conference on availability, reliability and security*. IEEE, 2011, pp. 307–312.

[46] V. Roussev and S. L. Garfinkel, "File fragment classification-the case for specialized approaches," in *2009 Fourth international IEEE workshop on systematic approaches to digital forensic engineering*. IEEE, 2009, pp. 3–14.

[47] C. E. Shannon, "A mathematical theory of cryptography," *Mathematical Theory of Cryptography*, 1945.

[48] J. Soto and J. Soto, *Randomness testing of the advanced encryption standard candidate algorithms*. US Department of Commerce, Technology Administration, National Institute of . . . , 1999.

[49] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[50] A. Lempel and J. Ziv, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[51] I. Kaplansky, "A common error concerning kurtosis." *Journal of the American Statistical Association*, 1945.

[52] P. H. Westfall, "Kurtosis as peakedness, 1905–2014. rip," *The American Statistician*, vol. 68, no. 3, pp. 191–195, 2014.

[53] P. Welch, "The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[54] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *digital investigation*, vol. 6, pp. S2–S11, 2009.

[55] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Napierone: A modern mixed file data set alternative to govdocs1," *Forensic Science International: Digital Investigation*, vol. 40, p. 301330, 2022.

[56] OpfLabs, "Openpreserve/format-corpus: An openly-licensed corpus of small example files, covering a wide range of formats and creation tools."

[57] N. Coldwell, "Comparison of audio compression." [Online]. Available: http://nigelcoldwell.co.uk/audio/

[58] "Digital corpora." [Online]. Available: https://digitalcorpora.org/corpora/files

[59] School of Computing at Edinburgh Napier University, "Napierone dataset." [Online]. Available: https://registry.opendata.aws/napierone/